Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise de projetos no GitHub que utilizam Behavior Driven Development

Tiago de Oliveira Kfouri

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Profa. Dra. Genaína Rodrigues

Brasília
2019

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise de projetos no GitHub que utilizam Behavior Driven Development

Tiago de Oliveira Kfouri

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Profa. Dra. Genaína Rodrigues (Orientadora)
CIC/UnB

Prof. Dr. Fernando Albuquerque
CIC/UnB

Profa. Dra. Fernanda Lima
CIC/UnB

Prof. Dr. Edison Ishikawa
Coordenadora do Bacharelado em Ciência da Computação

Brasília, 19 de julho de 2019

Dedicatória

Dedico este trabalho aos meus pais, que me educaram a ser quem sou hoje e aos meus amigos mais próximos, que sempre me apoiaram e me ajudaram quando eu precisava.

Agradecimentos

Agradeço à minha orientadora, Professora Doutora Genaína Nunes Rodrigues, que me guiou neste árduo trabalho, me mostrando o caminho certo para fazer um bom trabalho. Agradeço imensamente ao aluno de mestrado, Rafael Fazzolino, que me ajudou bastante estando sempre presente. Sem ele, esse trabalho não seria possível.

Agradeço também aos meus amigos de curso, que trilharam a mesma jornada e, por isso, sempre nos ajudávamos. Em especial ao George Geonardo, que começou a realizar este trabalho comigo, mas teve que se desvincular por motivos de força maior.

Agradeço aos professores da UnB, que me permitiram ter o conhecimento necessário para fazer este trabalho. Ao coordenador do curso, que resolveu muitos dos meus problemas.

Por fim, agradeço aos meus amigos mais próximos, que me apoiaram até o fim.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Este trabalho tem como objetivo extrair informações preliminares sobre a utilização da abordagem Behavior Driven Development (BDD) a partir da mineração de repositórios de software livre. Essas informações também podem ser usadas para melhoramento do uso de BDD nos projetos atuais. Além disso, esse trabalho pode ser usado para alavancar novos questionamentos futuros sobre como o BDD é usado.

Palavras-chave: BDD, Mineração de Repositórios de Software

Abstract

This study had the purpose of extracting preliminary information about the usage of the Behavior Driven Development (BDD) method by using data mining of open source repositories. With this knowledge, projects that already use BDD can improve their already used methods. Also, this work can be used for new future questioning about the usage of BDD.

Keywords: BDD, Software Repository Mining

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API Application Programming Interface.

BDD Behavior Driven Development.

GQM Goal Question Metric.

HTML Hypertext Markup Language.

HTTP HyperText Transfer Protocol.

IIQ Intervalo Interquartil.

JSON JavaScript Object Notation.

KDD Knowledge Discovery in Databases.

MER Modelo Entidade Relacionamento.

OSS Open Source Software.

TDD Test Driven Development.

Capítulo 1

Introdução

1.1 Contexto

Os métodos ágeis são uma reação aos métodos tradicionais de desenvolvimento de *software*, onde é reconhecida uma alternativa para processos de desenvolvimento de *software* dirigidos à documentação [6]. Tanto os métodos ágeis quanto os métodos dirigidos a planejamento possuem suas características próprias e existem projetos onde certo método claramente funciona melhor e o outro método terá dificuldades [7]. Segundo Beck [6], o primeiro princípio do manifesto ágil é "Nossa maior prioridade é satisfazer o cliente através de rápidas e contínuas entregas de *software*".

O Test Driven Development (TDD), inventado por Kent Back em 2003 [8], é um método ágil e consiste na criação automatizada de testes em pequenas e rápidas iterações, antes mesmo de se desenvolver qualquer código funcional. Visando melhorar o já existente TDD, Dan North inventou, em 2006, o processo chamado Behavior Driven Development (BDD) [9], que consiste em um conjunto de práticas de engenharia de *software* projetadas para ajudar equipes a construírem e entregarem *software* de melhor qualidade mais rápido [2].

Com o advento de abordagens como BDD, o processo de desenvolvimento de *software* pode evoluir e se adaptar. Neste sentido, diversas estratégias, técnicas e ferramentas são definidas com o objetivo de apoiar o processo de desenvolvimento seguindo abordagens como o BDD. Entretanto, para isso, vê-se a necessidade de compreender como a comunidade de desenvolvimento de *software* utiliza tais abordagens, maximizando a possibilidade da aderência da comunidade na utilização de novas estratégias, técnicas e ferramentas baseadas no uso do BDD.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral descobrir como o BDD é utilizado em repositórios Open Source Software (OSS), em particular, no *GitHub*. Ele também procura descobrir quais as linguagens mais relevantes que utilizam BDD e o foco com que a abordagem é dada ao longo do processo de desenvolvimento de *software*.

1.2.2 Objetivos Específicos

Essa pesquisa procura responder quais as características quantitativas relativas a tamanhos médios de *steps* por cenário e de cenários por *feature*. Em geral, a literatura recomenda que um cenário BDD deve ter em torno de 3 a 8 passos (*steps*) por cenário. Queremos avaliar se essa recomendação é seguida. Queremos também avaliar quais são as linguagens de programação mais utilizadas por repositórios que utilizam BDD. E por fim, planejamos saber com que frequência são feitas as atualizações BDD em relação aos *commits* do projeto.

1.3 Metodologia

Este trabalho é um estudo de caso e foi feito em quatro partes: definição do problema, planejamento, execução do estudo e análise dos dados.

1.4 Resultados Alcançados

Ao final desse trabalho, concluiu-se que as linguagens mais utilizadas em projetos que usam BDD são *Java*, *JavaScript*, *Python* e *Ruby*, que o número médio de *steps* por cenário é de 2 a 4, que o número médio de cenários por *feature* é de 1 a 5 e que não há relação direta entre o tamanho do repositório e seu número de *features*.

1.5 Organização do Trabalho

Este trabalho está organizado em cinco capítulos. No Capítulo 1, é apresentada uma introdução do trabalho. No Capítulo 2, temos o referencial teórico, que aborda brevemente sobre os conceitos de BDD e mineração de dados. No Capítulo 3, a metodologia é apresentada. Nela, é exibida a definição do problema, o planejamento e a execução da

pesquisa. No Capítulo 4, os dados são analisados. Por fim, no Capítulo 5, a conclusão da análise é apresentada.

Capítulo 2

Referencial Teórico

Este capítulo pretende expôr a base teórica do trabalho para que o leitor tenha conhecimento sobre o assunto tratado neste trabalho. Ele irá explicar melhor sobre o Behavior Driven Development na Seção 2.1, sobre mineração de dados na Seção 2.2 e sobre a API do *GitHub* na Seção 2.3.

2.1 Behavior Driven Development

O Behavior Driven Development (BDD) é um conjunto de práticas de engenharia de *software* projetadas para ajudar times a construírem e entregarem *software* de melhor qualidade [2]. Ele faz isso provendo uma linguagem comum baseada em sentenças simples e estruturadas expressas em linguagem *Gherkin*. Linguagem *Gherkin* é uma linguagem familiar, estruturada acerca do domínio de modelo, compartilhada entre o negócio e os *stakeholders* para comunicar as tarefas conectadas ao desenvolvimento do *software* [10].

O BDD foi originalmente inventado por Dan North, em 2006, visando melhorar o já existente Test Driven Development (TDD) [9]. O TDD, inventado por Kent Back em 2003 [8], consiste na criação automatizada de testes em pequenas e rápidas iterações, antes mesmo de se desenvolver qualquer código funcional. Depois, o código é refatorado até que o teste passe [11]. Para Smart [2], essa técnica é simples, porém poderosa, pois encoraja os desenvolvedores a escreverem códigos melhor estruturados e mais fáceis de se fazer manutenção, o que resulta em menos defeitos durante o processo. Apesar das vantagens do TDD, muitas vezes ele acaba fazendo o desenvolvedor ficar muito focado nos detalhes e perder a visão geral do projeto. Isso pode trazer alguns prejuízos ao projeto. Um deles é a criação de código inútil, por exemplo.

A Figura 2.1 demonstra como funciona o desenvolvimento usando a técnica de TDD. Primeiro, é escrito um exemplo do que o código deveria fazer, na forma de um teste que vai falhar. Depois, é escrito código apenas necessário para que o teste passe. Finalmente,

é revisado o código para ver se há algo que possa ser melhorado e, a partir disso, o código é refatorado.

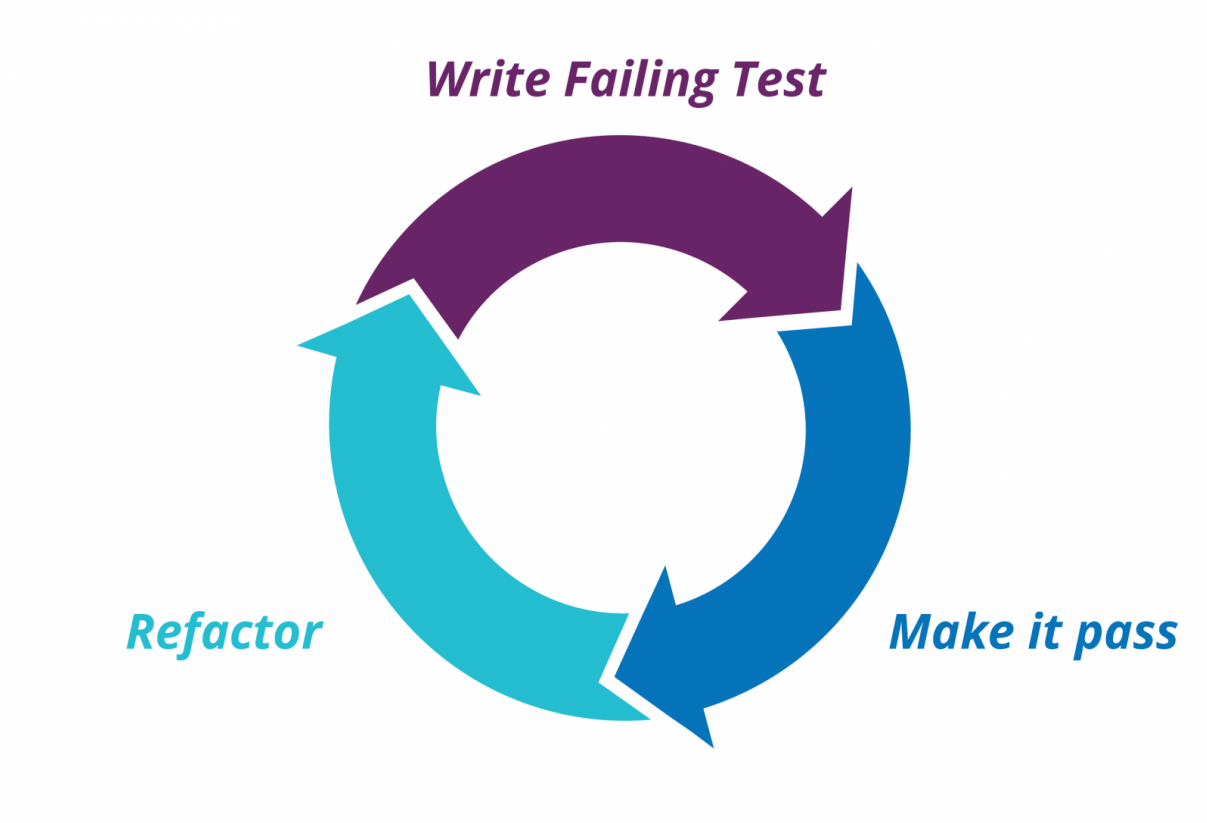


Figura 2.1: Esquema de funcionamento do TDD [1]

Segundo Smart [2], o principal foco do BDD é de ajudar a criar o código certo e de maneira certa. Um código que não é escrito de maneira certa, irá possuir erros, será difícil de fazer manutenção e, no geral, será descartado para uso, pois não atende seus requisitos. Já um código que não é certo é um código que não contribui para o objetivo do projeto. Ele é considerado como esforço inútil, pois recursos foram gastos nele e ele não será usado. Esses fatores parecem triviais, porém eles acontecem com grande frequência na indústria de *software*. O Chaos Report de 2008 [12] revelou que 42% dos projetos foram entregados atrasados, esgotaram seus recursos ou falharam em entregar todas as funcionalidades solicitadas e 21% dos projetos foram cancelados inteiramente.

A maioria das ferramentas que usam o BDD, como o *Cucumber* [13] e o *JBehave* [9], por exemplo, utilizam um formato de linguagem chamado *Gherkin*. O *Gherkin* é uma linguagem estruturada [10], sendo facilmente entendível pelos envolvidos e, ao mesmo tempo, fácil de se automatizar. Por isso, ele é útil tanto para documentação de especificações quanto para executar testes automatizados [2]. No *Gherkin*, os requisitos para uma funcionalidade são agrupados em um arquivo texto de extensão ".feature". Esse arquivo

possui uma curta descrição da funcionalidade, seguido de cenários ou exemplos de como a funcionalidade deve se comportar [13]. Podemos ver um exemplo desse tipo de arquivo na Figura 2.2.

Como pode ser visto, os requisitos do *Gherkin* estão expressos em inglês, mas como uma estrutura especificada. Cada *scenario* é feito por um número de *steps*, onde cada *step* começa com uma palavra chave (*Given, When, Then, And* e *But*).

A ordem natural do *scenario* é *Given ... When ... Then.* As palavras chave *And* e *But* podem ser usadas para juntar vários *Given, When* ou *Then* de um jeito mais fácil de ler [2].

```
Feature: Transferring money between accounts
  In order to manage my money more efficiently
  As a bank client
  I want to transfer funds between my accounts whenever I need to

  Scenario: Transferring money to a savings account
    Given my Current account has a balance of 1000.00
    And my Savings account has a balance of 2000.00
    When I transfer 500.00 from my Current account to my Savings account
    Then I should have 500.00 in my Current account
    And I should have 2500.00 in my Savings account
  Scenario: Transferring with insufficient funds
    Given my Current account has a balance of 1000.00
    And my Savings account has a balance of 2000.00
    When I transfer 1500.00 from my Current account to my Savings account
    Then I should receive an 'insufficient funds' error
    Then I should have 1000.00 in my Current account
    And I should have 2000.00 in my Savings account
```

Figura 2.2: Exemplo de uma funcionalidade no *Gherkin* [2]

Na Figura 2.3 podemos ver como o BDD agrega ao TDD. O BDD constrói, primeiramente, um *feature* explicando como aquela funcionalidade inteira deve se comportar, em forma de um teste que vai falhar. Depois, são feitos n ciclos de TDD até que o teste do *feature* passe. Finalmente, há a refatoração do *feature* para ver se há como melhorá-lo e o ciclo é recomeçado.

Nair [3] explica as diferenças entre o BDD e o TDD. No Test Driven Development, o teste é escrito para checar a implementação da funcionalidade, porém, conforme o código evolui, testes podem dar falsos resultados, pois a estrutura do projeto foi modificada. O Behavior Driven Development também possui uma abordagem inicial de testes, mas difere por testar o comportamento do sistema da perspectiva do usuário final. Por fim, no BDD, um teste é escrito que possa satisfazer tanto o desenvolvedor quanto o cliente, mas no TDD, o teste é feito apenas para satisfazer o desenvolvedor e o código que ele escreve.

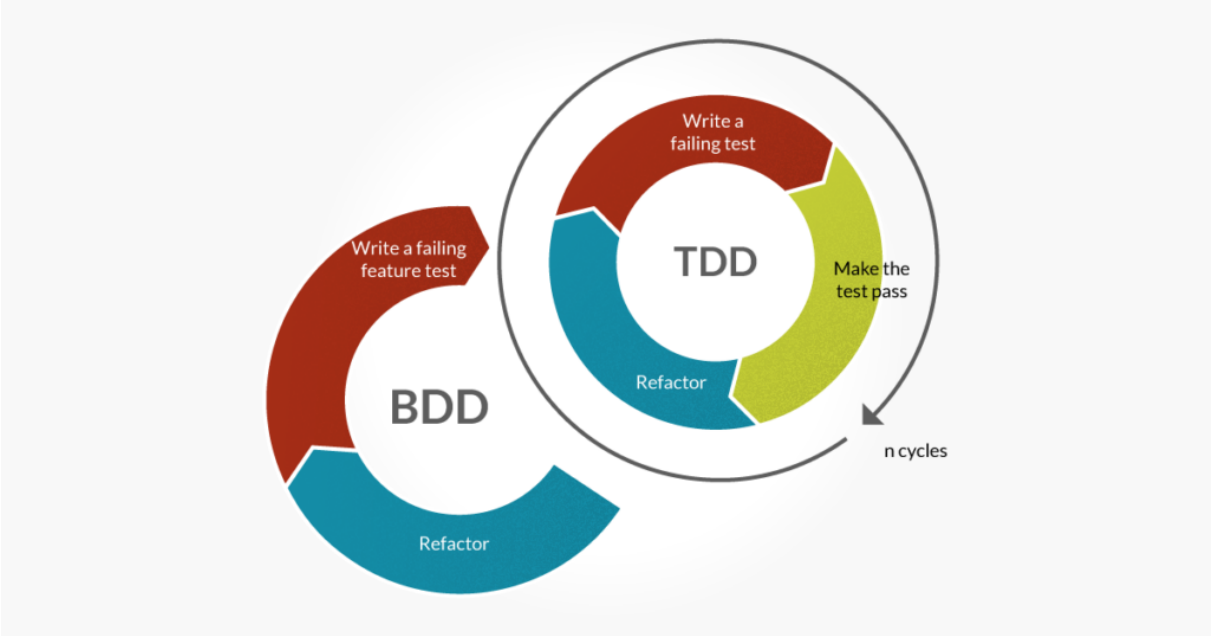


Figura 2.3: Esquema de funcionamento do BDD [3]

Assim como apontado por Vieira [14], nós ficamos sabendo de equipes que usam BDD em seu projeto, mas quando checamos, eles estão usando apenas uma ferramenta de BDD para um teste de automação e não o conceito de BDD em si. Segundo Agilealliance [15], o uso de BDD não precisa de nenhuma ferramenta ou linguagem de programação. Ele é primariamente uma abordagem conceitual. Transformá-lo em uma prática técnica ou que necessite de ferramentas específicas tiraria o ponto principal. Logo, Vieira [14] diz que a ideia principal por trás do BDD é que ele foi feito para prevenir problemas de comunicação, fazendo todos na equipe se comunicarem mais frequentemente, de maneira mais eficiente e usando exemplos da vida real e não usando abstrações.

Segundo Smart [2], um desenvolvedor que utiliza BDD não irá pensar em termos de escrever testes unitários para uma classe particular e sim de escrever especificações técnicas descrevendo como a aplicação deve se comportar em determinada situação.

Para finalizar, Smart [2] trás as vantagens e desvantagens do uso da técnica BDD. Os benefícios são os seguintes.

- Reduz desperdício, pois diminui problemas de comunicação;
- Reduz custos, que é uma consequência direta da redução de desperdício;
- É fácil e seguro de fazer mudanças, pois a documentação é acessível para todos os envolvidos com o projeto;
- Proporciona lançamentos mais rápidos, pois os testes automatizados agilizam o ciclo de lançamentos.

As desvantagens são listadas a seguir.

- Requer alto engajamento e colaboração, pois as técnicas BDD são baseadas em conversação e *feedback*;
- Funciona melhor em um contexto ágil, uma vez que os requerimentos irão evoluir conforme a equipe aprende mais sobre o projeto;
- Testes mal escritos podem levar a custos maiores quanto a manutenção de testes.

2.2 Mineração de Repositórios de Software

Segundo Hand [16], mineração de dados é a análise de dados observacionais para encontrar relações não esperadas e para resumir os dados em novas maneiras que sejam tanto entendíveis como úteis para o dono dos dados.

A motivação por trás da mineração de dados é que existem grandes bancos de dados que contêm informações que são de valor, mas essas informações estão escondidas dentre o número massivo de dados não interessantes e têm que ser descobertas. Isso é, uma pessoa está atrás dessa valiosa informação e o objetivo é extraí-la [17].

A análise de grandes quantidades de dados pelo homem é inviável sem o auxílio de ferramentas computacionais apropriadas. Portanto, torna-se imprescindível o desenvolvimento de ferramentas que auxiliem o ser humano, de forma automática e inteligente, na tarefa de analisar, interpretar e relacionar esses dados. Tudo isso para que se possa desenvolver e selecionar estratégias de ação em cada contexto de aplicação [18].

Segundo Fayyad [4], o Knowledge Discovery in Databases (KDD) é uma tentativa de solucionar o problema causado pela chamada "era da informação": a sobrecarga de dados. O KDD refere-se a todo o processo de descoberta de conhecimento, e a mineração de dados a uma das atividades do processo. Na Figura 2.4 podemos ver uma representação do processo de KDD.

Segundo Fayyad [4], as etapas do KDD são: selecionar um conjunto de dados nos quais a pesquisa será feita, limpar e pré-processar os dados para retirar ruídos, reduzir os dados, utilizar algum método de mineração de dados de acordo com os objetivos do KDD e, por último, interpretar os resultados obtidos.

De uma maneira geral, a complexidade do processo de KDD está na dificuldade em perceber e interpretar adequadamente inúmeros fatos observáveis durante o processo e na dificuldade em conjugar dinamicamente tais interpretações de forma a decidir quais ações devem ser realizadas em cada caso [18].

Para Fayaad [4], os objetivos do KDD são definidos pela intenção de uso do sistema. Podemos distinguir dois tipos de objetivos:

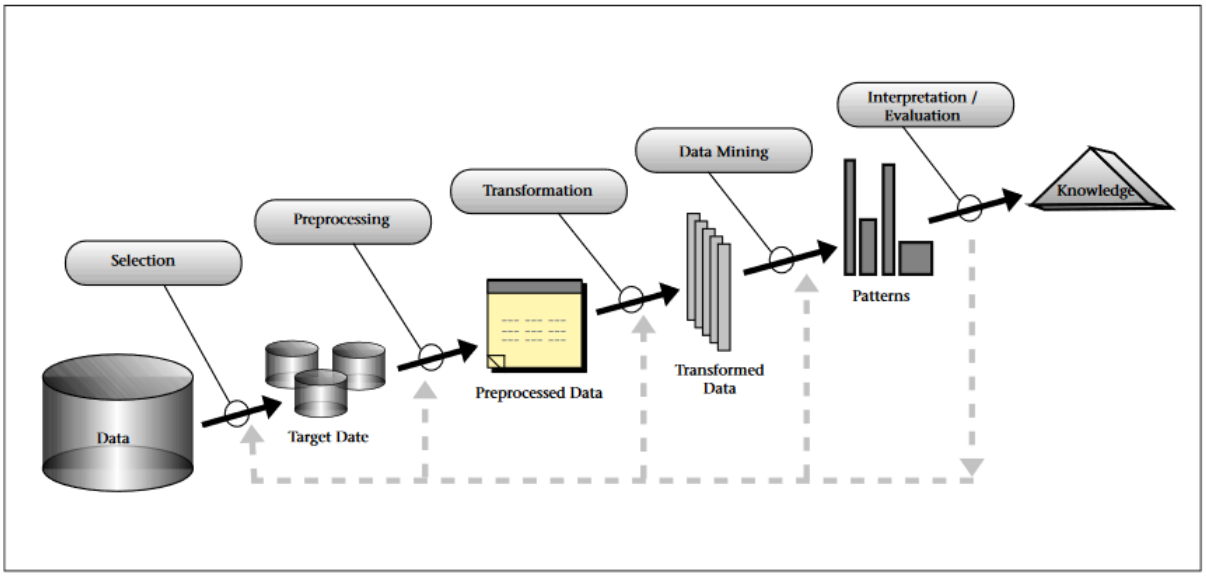


Figura 2.4: Representação do processo KDD [4]

1. **Verificação**: o sistema é limitado para a verificação das hipóteses do usuário;
2. **Descoberta**: o sistema autonomamente descobre novos padrões.

Um dos desafios da mineração de dados é a contaminação dos dados, ou seja, certos dados entre a amostra não refletem a maioria e podem acabar modificando os resultados das análises destes dados. Segundo Hand [17], provavelmente não é exagero dizer que todos os meios de conjuntos de dados são contaminados, apesar de que em conjuntos de dados pequenos, isso possa ser difícil de detectar. Com conjuntos de dados grandes, isso significa que o minerador de dados pode encontrar padrões incomuns, que são simplesmente um artefato do conjunto de dados, registro ou outras inadequações.

2.3 GitHub REST API v3

Segundo Kalliamvakou [19], o *GitHub* é um site que hospeda códigos de uma maneira colaborativa. Ele já possui mais de 10 milhões de repositórios e está se tornando uma das fontes de maior importância no quesito de artefatos de *software* na internet.

O *GitHub* possui uma Application Programming Interface (API) que auxilia na coleta de dados sobre repositórios hospedados em seu site. Essa API se chama "GitHub REST API v3" e, segundo sua documentação [20], funciona através de requisições HyperText Transfer Protocol (HTTP) para o seguinte endereço:

```
1 https://api.github.com
```

Toda informação que é enviada e recebida está no formato JavaScript Object Notation (JSON), campos vazios são incluídos como *null* ao invés de serem omitidos e dados relativos a tempo são retornados no seguinte formato:

```
YYYY-MM-DDTHH:MM:SSZ
```

Ao se fazer uma requisição do tipo GET, especificando um repositório, é possível recuperar todas as informações referentes a dado repositório. Essa requisição é feita no formato a seguir, onde ":owner" é o nome do dono do repositório e ":repo" é o nome do repositório:

```
https://api.github.com/repos/:owner/:repo
```

A seguir, podemos visualizar uma parte de um exemplo de retorno a esse tipo de requisição. Para este exemplo, foi requisitado o repositório de nome "mocha", cujo dono possui nome "mochajs".

```
{
"id": 1451352,
  "node_id": "MDEwOlJlcG9zaXRvcnkxNDUxMzUy",
  "name": "mocha",
  "full_name": "mochajs/mocha",
  "private": false,
  "owner": {
    "login": "mochajs",
    "id": 8770005,
    "node_id": "MDEyOk9yZ2FuaXphdGlvbjg3NzAwMDU=",
    "avatar_url": "https://avatars2.githubusercontent.com/u/8770005?v=4"
   ,
    "gravatar_id": "",
    "url": "https://api.github.com/users/mochajs"
    }
}
```

Kalliamvakou [19] declara que a maior ameaça para a validação de qualquer estudo que usa o *GitHub* é que a ferramenta é usada indiscriminadamente para uso pessoal. Enquanto muitos repositórios estão ativamente sendo desenvolvidos no *GitHub*, a maioria deles são repositórios pessoais e inativos. Assim sendo, uma das perguntas mais importantes a se considerar quando estiver usando dados do *GitHub* é a de qual tipo de repositório está sendo estudado e, com essa resposta, é possível escolher repositórios adequados para amostra.

Capítulo 3

Metodologia

A metodologia usada neste trabalho é a de estudo de caso, que é um estudo de observação [21]. Nele, é conduzida uma investigação de uma entidade ou fenômeno durante um período de tempo [22]. Um estudo de caso pode ser classificado tanto como quantitativo quanto como qualitativo, dependendo do objetivo da investigação [23]. No caso desta pesquisa, ele é inteiramente quantitativo.

Este capítulo irá explicar toda a metodologia usada para o desenvolvimento deste estudo de caso.

3.1 Definição do Problema

Segundo Wholin [23], na fase de definição, a fundação do projeto é determinada. Se a fundação não for criada devidamente, retrabalho pode ser necessário, ou pior, o projeto não pode ser usado para estudar o que foi pretendido. O propósito da fase de definição é definir os objetivos de um estudo de caso de acordo com uma estrutura definida.

A estrutura usada para definir o estudo de caso pode ser encontrada na Tabela 3.1. Ela é uma versão modificada de um modelo Goal Question Metric (GQM) e foi proposta por Briand [24].

3.1.1 Objeto de Estudo

Para Yin [25], para se definir um caso, é preciso considerar dois diferentes passos: definição do caso e limitação do caso. Ao definir o caso, é pensado no objeto a ser estudado. Muitas vezes, o caso é definido como sendo uma pessoa, porém ele também pode ser uma entidade ou evento. Após a definição, deve-se limitar o caso. Para fazer isso, deve-se distinguir o que é importante a ser estudado sobre o caso e o que não é importante. Assim, você consegue delimitar quais casos são relevantes para o seu estudo. Com esses dois passos

Dimensão	Definição	Exemplos
Objeto de Estudo	O que será analisado	Processo de desenvolvimento, teste de sistema, produto final
Propósito	O porquê do objeto ser analisado	Monitoramento, controle, mudança
Foco de Qualidade	Qual propriedade/atributo do objeto será analisada	Custo, facilidade de uso, confiabilidade
Ponto de Vista	Quem usará os dados coletados	Líder do projeto, desenvolvedores, gerente de projeto
Contexto	O ambiente	Projeto X, na corporação A

Tabela 3.1: modelo usado para definição do estudo de caso

completos, fica mais fácil determinar o escopo da coleta de dados e, em particular, como distinguir dados do sujeito pesquisado no caso de estudo de dados externos ao caso.

Para essa pesquisa, o caso definido são projetos de *software* que utilizam o BDD. Quanto à limitação do caso, foram escolhidos apenas projetos Open Source Software (OSS) do *Github*.

3.1.2 Propósito

O propósito deste Estudo de Caso é fazer uma análise quantitativa do uso de BDD em projetos OSS no *GitHub*. Sabendo isso, novos projetos podem seguir os moldes mais populares. Essas informações também podem ser usadas para melhoramento do uso de BDD nos projetos atuais.

3.1.3 Foco de Qualidade

Foco de qualidade é o efeito primário que está sendo estudado em um experimento [23]. Segundo Briand [24], o foco de qualidade abrange um atributo particular do objeto que será estudado, monitorado, controlado ou modificado. Ele é a fraqueza mais urgente que precisa ser analisada. Exemplos de foco de qualidade são custo, confiabilidade, corretude, mudanças, facilidade de uso e manutenibilidade.

Para este trabalho, queremos apenas explorar os repositórios públicos para aprofundar o entendimento do uso do BDD em projetos OSS, portanto não há um foco de qualidade definido.

3.1.4 Perspectiva

Também segundo Briand [24], o ponto de vista identifica as funções ou posições das pessoas que usarão os resultados das análises dos dados. O objeto de estudo será pesquisado a

partir do ponto de vista de engenheiros de *software*, que são as pessoas que utilizam o BDD.

3.1.5 Contexto

Para Wholin [23], contexto é o ambiente em que o experimento acontece, ou seja, aonde se encontra o objeto de estudo. Não devemos confundir, porém, o contexto com o objeto de estudo. Enquanto o objeto de estudo são os projetos em si, o contexto é aonde o objeto de estudo está inserido e tudo que o influencia. Neste estudo, os repositórios estão salvos no *GitHub* e são Open Source Software.

3.2 Planejamento do Estudo de Caso

Nessa seção, será ilustrado como o estudo de caso foi planejado. Esse planejamento consiste nas seguintes partes:

- Formulação de Proposições: as proposições são declaradas explicitamente e os dados obtidos durante o curso do estudo de caso são usados para responder a estas proposições;
- Seleção de Variáveis: escolha dos dados que serão buscados durante a pesquisa para a resposta das proposições;
- Seleção de Sujeitos: escolha dos objetos a serem estudados;
- Forma de Análise: decisão de como analisar os dados da melhor maneira possível;
- Instrumentação: os instrumentos a serem utilizados para realizar o estudo de caso.

3.2.1 Formulação de Proposições

As proposições formuladas para este estudo de caso foram:

1. Quais as linguagens de programação mais usadas por repositórios que usam BDD?
2. Quantos *steps* têm em cada cenário?
3. Quantos cenários têm em cada *feature*?
4. O tamanho do repositório tem relação com o número de *features*?

Estas proposições servirão de guia para o estudo e todos os métodos de análise serão estabelecidos conforme a necessidade de respondê-las.

3.2.2 Seleção de Variáveis

Pensando em quais dados sobre o repositório seriam importantes, foi criado o Modelo Entidade Relacionamento (MER) apresentado na Figura 3.1, que representa o banco de dados que será usado nesta pesquisa. Além dos dados relevantes para esse trabalho, foram salvos também alguns dados que não foram usados, mas que poderiam ser importantes em algum trabalho futuro.

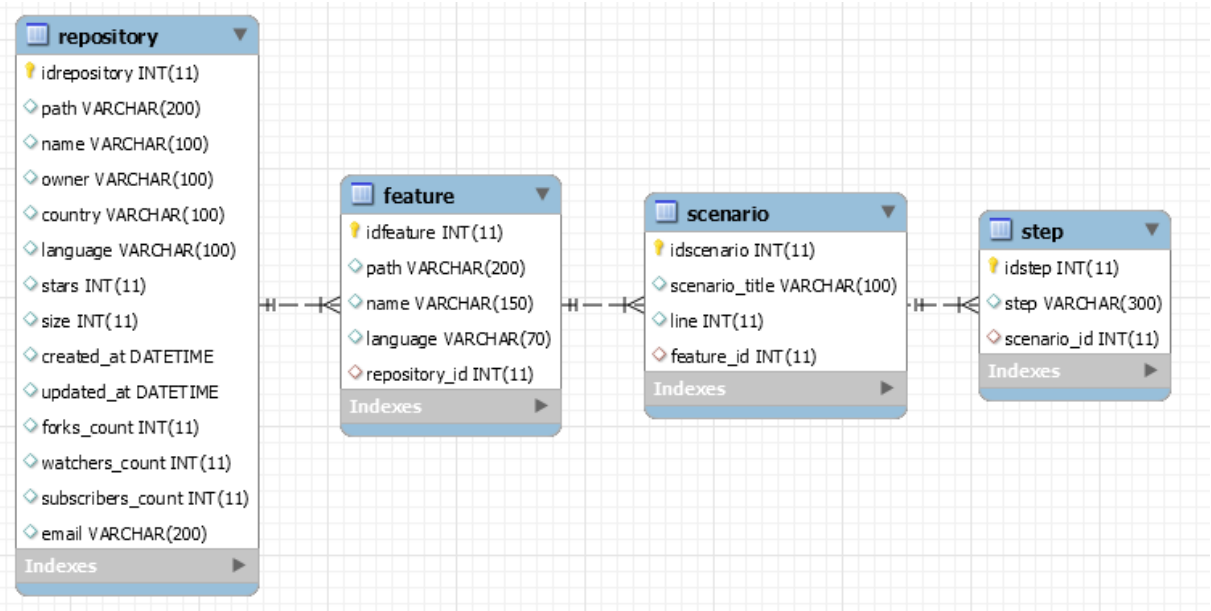


Figura 3.1: Modelo Entidade Relacionamento

O MER possui quatro entidades, representando um repositório, um *feature*, um cenário e um *step*. Quanto aos relacionamentos entre essas entidades, um repositório pode ter várias *features*, uma *feature* pode ter vários cenários e um cenário pode ter vários *steps*.

As variáveis que serão utilizadas para o estudo de caso são: número de *steps*, número de cenários, número de *features*, tamanho do repositório, linguagem de programação do repositório, número de estrelas do repositório, data do último *update* do repositório e número de *forks* do repositório.

3.2.3 Seleção de Sujeitos

Nesta etapa, é feita a escolha dos objetos a serem estudos, que são chamados de sujeito. O sujeito deste estudo de caso é o próprio caso, definido na Seção 3.1.1, ou seja, repositórios Open Source Software (OSS) que estão no *GitHub* e que utilizam BDD. O *GitHub* foi escolhido por ser uma plataforma aberta, popular e por possuir uma Application Programming Interface (API) que ajuda nessa coleta de dados. Foi definido, também, um

número mínimo de 150 projetos válidos. Para ser considerado válido, o projeto deve ter pouca diferença entre o último *update* do BDD e o último *update* do código fonte, pois os repositórios que possuem essa diferença muito alta são repositórios que começaram a utilizar BDD no início de seu desenvolvimento e pararam de utilizar em algum ponto. Assim, estes projetos não utilizam BDD de verdade e não devem ser considerados como válidos. Além disso, o projeto deve ter, no mínimo, três estrelas, para que o estudo tenha certa relevância e não considere projetos muito pequenos. Por último, foi definido que os projetos seriam procurados usando uma *tag*, no caso BDD, como pesquisa e ordenando por número de estrelas.

3.2.4 Forma de Análise

Para se descobrir quantos *steps* têm em cada cenário e quantos cenários têm em cada *feature*, foi decido usar *boxplot*, pois ele permite analisar e comparar a variação de uma variável entre diferentes grupos de dados, ou seja, nos permite ver onde estão os valores mais prováveis, além de valores extremos. Um exemplo de *boxplot* pode ser visto na Figura 3.2

O intervalo entre o primeiro quartil (Q1, ou quartil inferior) e o terceiro quartil (Q3, ou quartil superior) é chamado de Intervalo Interquartil (IIQ) e representa onde a maioria dos dados de encontra. Os limites inferior e superior representam o valor mínimo e máximo, respectivamente. O limite inferior é calculado como sendo Q1 - 1.5 x IIQ e o limite superior é calculado como sendo Q3 + 1.5 x IIQ. A mediana é o valor exatamente no centro do conjunto de dados ordenado em ordem crescente. Por fim, os discrepantes são os valores fora do limite inferior ou superior.

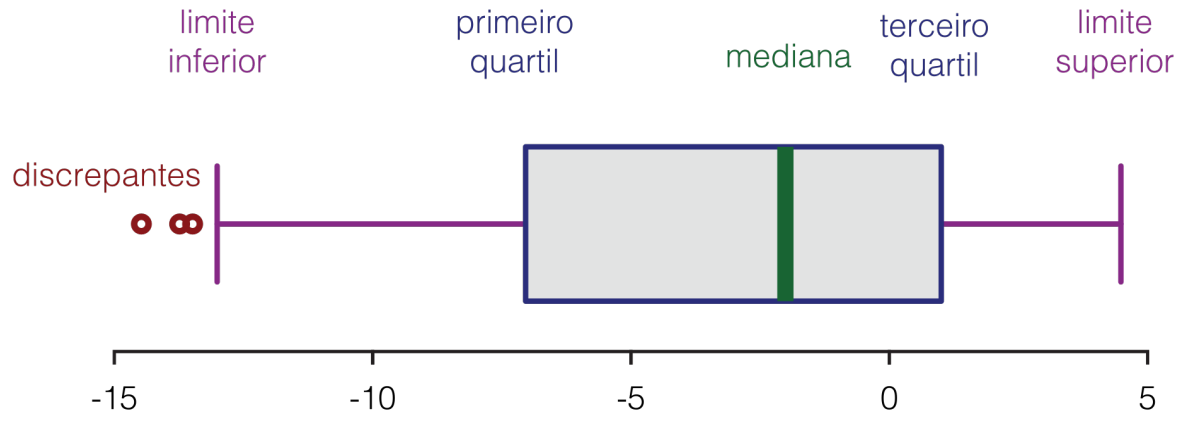


Figura 3.2: Exemplo de *boxplot* [5]

Para se descobrir se o tamanho do repositório tem relação com o número de *features*, foi decidido que seria feita uma análise através do coeficiente de correlação de Pearson.

Diferentemente das análises anteriores, essa proposição não foi analisada com gráficos, pois, mesmo existindo gráficos capazes de mostrar o tamanho do repositório em função do seu número de *features*, eles não são suficientes para se chegar a uma análise conclusiva sobre as duas variáveis terem correlação.

Segundo [23], o coeficiente de Pearson é um valor entre -1 e 1 e, se não há correlação entre as variáveis, ele é 0. Esse coeficiente r é calculado a partir da seguinte fórmula:

$$r = \frac{(n \cdot \sum_{i=1}^{n} x_i y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{\sqrt{(n \cdot \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2) \cdot (n \cdot \sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2)}}$$

onde $x_1, x_2, ..., x_n$ e $y_1, y_2, ..., y_n$ são os valores medidos de cada uma das variáveis.

Isso vai nos permitir descobrir se há uma correlação entre o tamanho do repositório e o número de *features*.

Para essa análise, também serão feitos gráficos de linha para termos uma visualização geral de como os dados estão dispostos, porém o fator definitivo para validade desse ponto é o coeficiente de correlação de Pearson.

Finalmente, para saber quais as linguagens de programação mais usadas por repositórios que usam BDD, foi utilizado um gráfico de barras, visto que ele representa bem uma quantidade de itens de acordo com uma categoria, que é o que procuramos. Podemos verificar um exemplo de um gráfico de barras na Figura 3.3. O gráfico de barras possui uma categoria em um de seus eixos, no caso do exemplo um certo hospital, e no outro eixo, o número de amostras sobre a categoria, no caso do exemplo o número de nascimentos. Para esse estudo, este tipo de gráfico possuirá as linguagens de programação dos repositórios como categoria em um dos eixos e o número de repositórios que utiliza determinada linguagem de programação no outro eixo.

3.2.5 Instrumentação

Foi definido que a linguagem de programação utilizada do programa de mineração de dados seria *Python*, por ser uma linguagem de programação já conhecida pelo autor deste projeto e possuir bibliotecas que simplificam o trabalho. São elas:

- **Matplotlib**: biblioteca de construção de gráficos;
- **Sqlalchemy**: biblioteca para conexão com o banco de dados;
- **BeautifulSoup**: interpretador para Hypertext Markup Language (HTML);
- **Requests**: permite fazer requisições HyperText Transfer Protocol (HTTP);
- **Scipy**: usada para calcular o coeficiente de correlação de Pearson;

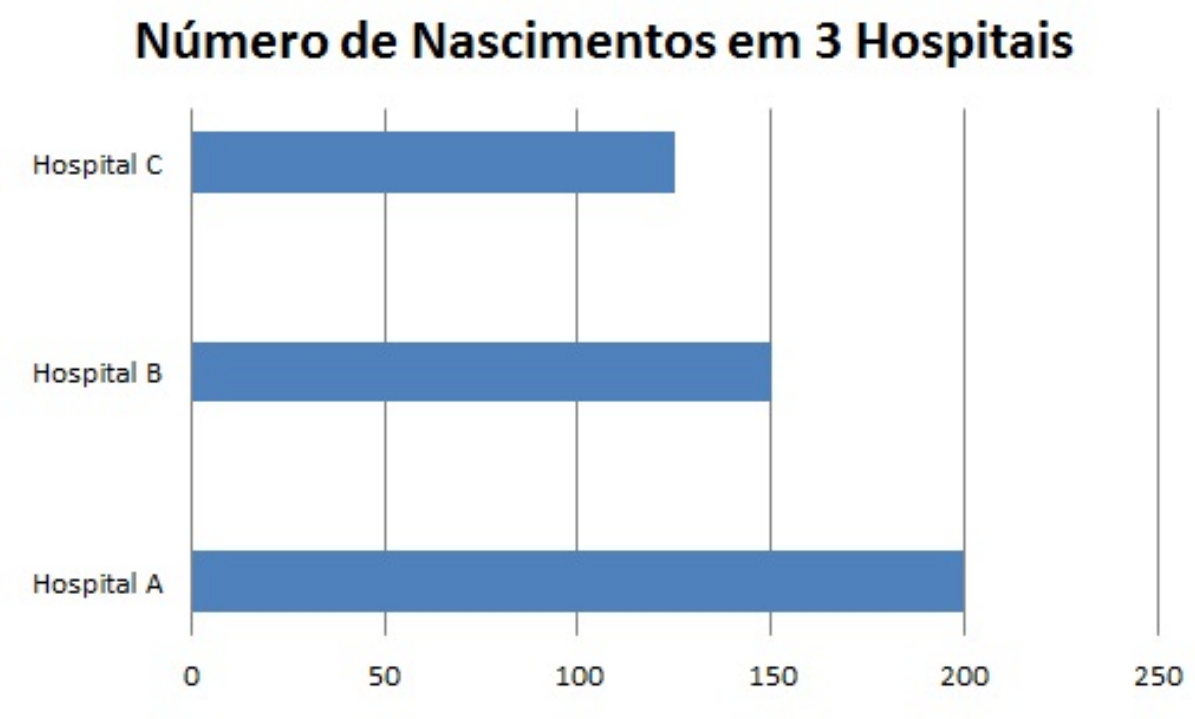


Figura 3.3: Exemplo de um gráfico de barras

- **Pandas**: biblioteca para organizar conjuntos de dados.

Ademais, foi utilizado um ambiente integrado de desenvolvimento denominado *PyCharm* para auxílio da programação e organização do projeto.

Também foi definido que o banco de dados a ser usado seria o *MySQL*, por ser um banco de dados popular, rápido e de fácil manipulação. Foi utilizado o programa *MySQL Workbench* para administração desse banco de dados.

O projeto foi esquematizado de acordo com a Figura 3.4, onde a camada "View" é responsável pela comunicação entre usuário e o programa, a camada "View_model" é responsável por conectar a "View" com o banco de dados, além de realizar todos os outros processamentos necessários e a camada "Model" é responsável por modelar os objetos que a camada "View_model" usará. Cada um desses objetos representa uma entidade do banco de dados. Os detalhes de como o projeto funciona serão explicados na Seção 3.3.

3.3 Execução do Estudo de Caso

Esta seção explicará detalhadamente os processos realizados nesse estudo de caso. Para a execução deste estudo de caso, foi criado um projeto OSS[1] que permitiu a coleta e análise

[1]https://github.com/ggpsgeorge/minerador

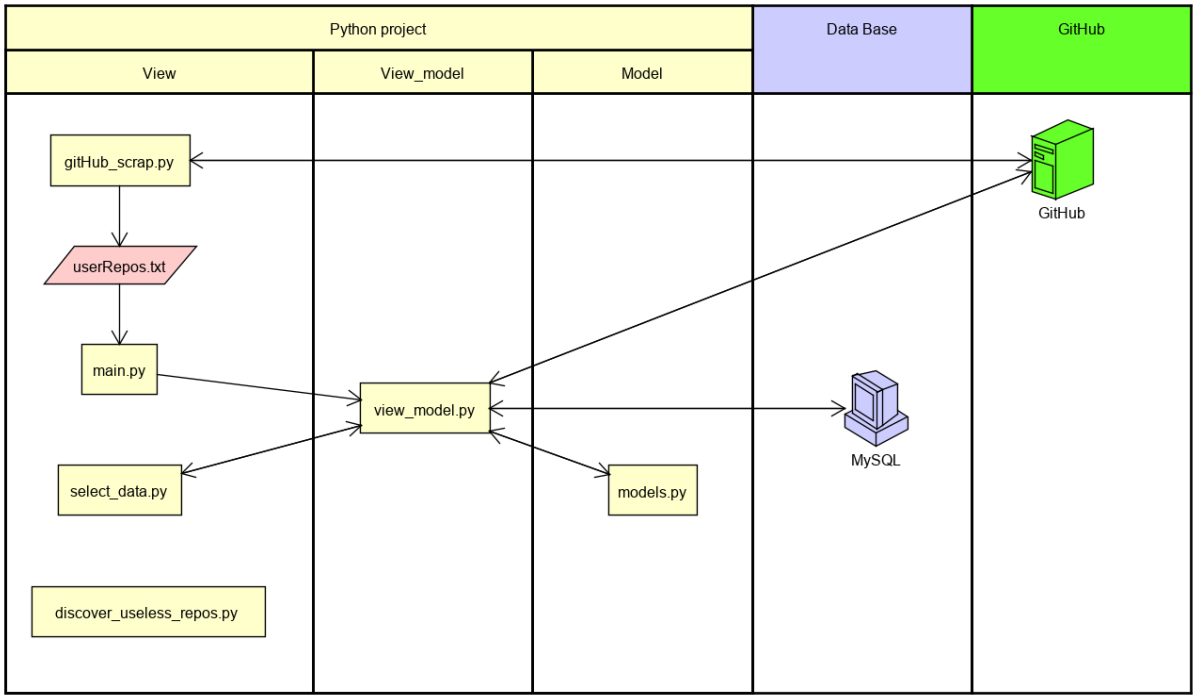


Figura 3.4: Esquema do projeto de mineração de repositórios BDD

dos dados necessitados para a pesquisa. Esse projeto encontra-se armazenado no *GitHub* e pode ser acessado por qualquer pessoa. Um esquema do projeto pode ser encontrado na Figura 3.4.

3.3.1 Busca dos Nomes dos Repositórios

Para sabermos se um repositório está usando a técnica de BDD, precisamos descobrir se há algum arquivo nele com a extensão ".feature". Porém, para não procurarmos aleatoriamente por repositórios que usam BDD entre os milhares de repositórios no *GitHub*, foi feito um programa simples chamado "gitHub_scrap.py", que procura no *GitHub* por nomes de repositórios usando uma *tag*, no caso BDD, como pesquisa e ordenando por número de estrelas.

O programa, ao ser iniciado, requer uma entrada do usuário, referente ao número da página inicial. Na primeira vez que o programa é iniciado, é esperado que o valor de entrada do usuário seja 1, para que o programa colete os dados a partir da primeira página. Isso é necessário porque a pesquisa retorna várias páginas de resultados. Caso aconteça algo inesperado e seja necessário terminar o programa no meio do seu processo, pode-se retornar de onde havia parado entrando com o valor da página desejado.

Após isso, o programa entra em um laço para buscar, a cada iteração, 9 páginas. A busca é feita até a página 80. No final de cada iteração, há um comando para esperar por

120 segundos, pois a API do *GitHub* aceita apenas pesquisas de 9 páginas por minuto. O código abaixo reflete o que foi abordado do código até o presente momento.

```
inipage = int(input("Ini: "))
finpage = inipage + 9

while(finpage < 81):

  time.sleep(120)

  inipage = finpage
  finpage = inipage + 9
```

Para cada iteração, o programa cria um arquivo "usersRepospX_Y.txt", onde X é a página inicial e Y é a página final. Como exemplo, o primeiro arquivo criado será chamado "userReposp1_9.txt".

Depois, para cada página, é criada uma solicitação dos nomes dos repositórios através de uma URL, demostrada a seguir, onde j é o número da página:

```
url = "https://github.com/search?o=desc&q=BDD&s=stars&type=Repositories&
    p=" + str(j)
```

A solicitação, então, é feita através de uma requisição HyperText Transfer Protocol (HTTP) para o *GitHub*, através da biblioteca "Requests", e o resultado é a página requisitada da busca dos projetos com a tag BDD por ordem descendente de estrelas. É usada, então, a biblioteca "BeautifulSoup" para interpretar o resultado e, assim, adquirir os nomes dos repositórios. Os nomes destes projetos são então salvos no arquivo correspondente criado anteriormente e o programa chega à sua conclusão ao fim de todas as suas iterações. Abaixo, pode-se ver o trecho de código onde essa requisição é feita, interpretada e o resultado é salvo no arquivo correspondente:

```
  for j in range(inipage, finpage):
    github = requests.get(url + str(j))
    soup = BeautifulSoup(github.text, "html.parser")

    all_results = soup.find_all("a", attrs={"class":"v-align-middle"},
    href=True)

    for result in all_results:
      f.write(result['href'] + "\n")
```

3.3.2 Modelagem dos Objetos do Banco de Dados

Para representar cada entidade do banco de dados, foi feita uma classe correspondente a cada uma dessas entidades. Assim, foram criadas as classes "Repository", "Feature",

"Scenario" e "Step". Cada uma dessas classes herda de uma classe chamada "Base", para que a biblioteca "Sqlalchemy" possa reconhecer essas classes e transformá-las em entidades para salvar no banco de dados. Também é necessário mapear a classe de acordo com sua entidade correspondente, como pode ser exemplificado no código seguinte, referente à classe "Repository":

```
class Repository(Base):

    __tablename__ = 'repository'

    idrepository = Column(Integer, primary_key=True)
    path = Column(String)
    name = Column(String)
    owner = Column(String)
    country = Column(String)
    language = Column(String)
    stars = Column(Integer)
    size = Column(Integer)
    created_at = Column(DateTime)
    updated_at = Column(DateTime)
    forks_count = Column(Integer)
    watchers_count = Column(Integer)
    subscribers_count = Column(Integer)
    email = Column(String)
    features = relationship("Feature", cascade="all, delete-orphan")
```

3.3.3 Coleta dos Repositórios e Armazenamento no Banco de Dados

A tarefa de coleta dos repositórios e armazenamento no banco de dados é feita inicializando o programa "main.py". Inicialmente, ele requer um *token* de identificação que pode ser adquirido manualmente pelo site do *GitHub*. Esse *token* é necessário, pois o *GitHub* normalmente só permite, no máximo, 60 requisições por hora do mesmo endereço de IP, quantidade insuficiente para as nossas necessidades. Com o *token*, a quantidade máxima de requisições aumenta para 5000 por hora, atendendo às nossas necessidades.

Depois, o programa abre um dos arquivos "userReposp.txt" criado anteriormente pelo programa "gitHub_scrap.py" e cria uma lista com os caminhos para cada repositório. A criação dessa lista pode ser vista no código a seguir:

```
arq = "userResposp1_9.txt"
f = open(arq, "r")
ls_users = f.read().splitlines()
f.close()
```

```
5
6 pathapi = "https://api.github.com/repos"
7
8 ls_paths = []
9 for user in ls_users:
10     ls_paths.append(pathapi + user)
```

Percebe-se que o caminho para o repositório é estruturado do seguinte formato:

```
1 "https://api.github.com/repos/(nome do usuario)/(nome do repositorio)"
```

Com a lista de caminhos para repositórios completa, o programa começa um laço que, para cada caminho da lista, faz uma requisição HTTP ao *GitHub*, recebe um JSON como resposta, contendo todas as informações do repositório, salva as informações pertinentes em objetos "Repository", "Feature", "Scenario" e "Step" e salva esses objetos no banco de dados. Esses passos estão indicados no código seguinte:

```
1 for path in ls_paths:
2     try:
3         print("Downloading repository from path: " + path)
4         repository = viewRepository.getRepositoryFromPath(path)
5         print("Saving repository " + path + " on BD")
6         if repository.features != None:
7             viewRepository.saveRepositoryOnDB(repository)
8     except:
9         problem_paths.append(path)
10        print("There was a problem with the repository from path: " +
    path)
```

Há um tratamento de erro durante esse processo e, caso haja algum problema com algum repositório, o programa salva o nome do repositório o qual possuiu erro para recuperação manual futura e continua seu processo. Também é importante notar que, mesmo tendo feito o primeiro filtro nos repositórios procurando por repositórios com a tag BDD, muitos deles não possuem nenhum arquivo de extensão ".feature", ou seja, não utilizam BDD. Assim, nessa etapa de coleta de dados, já são descartados os repositórios que não possuem nenhuma "feature", ou seja, repositórios que não possuem nenhum arquivo com extensão ".feature", pois estes não nos interessam, uma vez que não utilizam BDD.

A classe "viewRepository" está contida no arquivo "view_model.py" e possui os métodos para coleta dos repositórios e armazenamento no banco de dados, que são os métodos "getRepositoryFromPath(path)" e "saveRepositoryOnDB(repository)", respectivamente.

3.3.4 Aprofundamento no Método de Coleta de Repositórios

O método "getRepositoryFromPath(path)" recebe um caminho para um repositório do *GitHub* e então chama o método "get_json_requests(path)", que cria uma requisição HTTP para a API do *GitHub*, utilizando a biblioteca "Requests", passando o *token* na requisição para permitir mais requisições por hora. O *GitHub*, então, retorna um JSON com as informações do repositório. No código abaixo, podemos vizualizar o código do método "get_json_requests(path)":

```
def get_json_requests(self, url):
    resp = requests.get(url, headers={'Authorization': 'token {}'.format
   (self.token)})
    return resp.json()
```

A seguir, podemos ver uma parte de um exemplo do JSON retornado pelo *GitHub*. Neste exemplo, foi usado o repositório de nome "mocha", do usuário "mochajs".

```
{
"id": 1451352,
  "node_id": "MDEwOlJlcG9zaXRvcnkxNDUxMzUy",
  "name": "mocha",
  "full_name": "mochajs/mocha",
  "private": false,
  "owner": {
    "login": "mochajs",
    "id": 8770005,
    "node_id": "MDEyOk9yZ2FuaXphdGlvbjg3NzAwMDU=",
    "avatar_url": "https://avatars2.githubusercontent.com/u/8770005?v=4"
   ,
    "gravatar_id": "",
    "url": "https://api.github.com/users/mochajs"
    }
}
```

Com as informações do repositório, o programa começa a salvar os dados relevantes em um objeto "Repository", conforme o código a seguir:

```
repositoryJson = self.get_json_requests(path)
ownerJson = self.get_json_requests(repositoryJson['owner']['url'])
repository = Repository()
repository.path = repositoryJson['url']
repository.name = repositoryJson['name']
repository.owner = repositoryJson['owner']['login']
repository.country = ownerJson['location']
repository.language = repositoryJson['language']
repository.stars = repositoryJson['stargazers_count']
repository.size = repositoryJson['size']
```

```
repository.created_at = datetime.strptime(repositoryJson['created_at'],
    '%Y-%m-%dT%H:%M:%SZ')
repository.updated_at = datetime.strptime(repositoryJson['updated_at'],
    '%Y-%m-%dT%H:%M:%SZ')
repository.forks_count = repositoryJson['forks_count']
repository.watchers_count = repositoryJson['watchers_count']
repository.subscribers_count = repositoryJson['subscribers_count']
repository.email = ownerJson['email']
```

Após os dados do repositório terem sido salvos, o método "getRepositoryFromPath (path)" chama o método "download_files(self, url, dirname, extensao)", que faz uma busca recursiva por todas as pastas do repositório. Para cada pasta, ele faz uma requisição HTTP para a API do *GitHub* solicitando uma lista dos arquivos da pasta atual e, com a resposta, procura por arquivos com a extensão ".feature". Caso encontre, o programa baixa o arquivo localmente em uma pasta com o nome do repositório dentro da pasta "dados" do repositório de mineração. A seguir encontra-se o código desse método:

```
def download_files(self, url, dirname, extensao):
    dir_urls = []
    data = self.get_json_requests(url)
    for raw in data:
        print("Checking " + raw['name'])
        if self.find_ext(raw['name'], extensao):
            print("Downloading " + raw['name'])
            resp = requests.get(raw['download_url'], allow_redirects=
   True, headers={'Authorization': 'token {}'.format(self.token)})

            open("dados/" + dirname + "/" + raw['name'], 'wb').write(
   resp.content)

        if raw['type'] == "dir":
            dir_urls.append(raw['url'])

    if dir_urls != []:
        for dr in dir_urls:
            self.download_files(dr, dirname, extensao)
```

Com os arquivos "feature" salvos, o método, então, começa a lê-los, distinguindo seus "scenarios" e "steps" e salvando-os em objetos respectivos. Essa parte é feita pelo método "get_feature_information(path)", sendo "path" o caminho para o arquivo "feature" salvo localmente. Esse método pertence originalmente a um projeto OSS[2].

A parte do método "getRepositoryFromPath(path)" que procura pelos arquivos "feature", salva-os localmente e os lê pode ser encontrada no trecho de código a seguir:

[2]https://github.com/BDD-OperationalProfile/trace_feature

```
dirname = repository.owner + "_" + repository.name

os.mkdir("dados/" + dirname)

# now getting the projects features and saving in dirs
self.download_files(repository.path + '/contents', dirname, "feature")

# now reading the feature file and saving as an object in the repository
    object
features = os.listdir(os.getcwd() + os.sep + "dados" + os.sep + dirname)
repository.features = []

for feature in features:
    repository.features.append(self.get_feature_information(os.getcwd()
   + os.sep + "dados" + os.sep + dirname + os.sep + feature))
```

Após esse passo, o objeto "Repository" está totalmente preenchido com seus dados necessários, possuindo, inclusive, seus respectivos objetos "Feature", "Scenario" e "Step". O objeto "Repository", então, é retornado pelo método.

3.3.5 Aprofundamento no Método de Armazenamento no Banco de Dados

O método "saveRepositoryOnDB(repository)", que é responsável por armazenar o objeto "Repositório" no banco de dados, é bastante simplificado pelo uso da biblioteca "Sqlalchemy". Primeiramente, ele cria uma nova "Session", que está responsável por estabelecer e manter todas as conversas entre o programa e o banco de dados. Depois, é criado o "schema" do banco de dados, ou seja, ele cria as tabelas no banco de dados referentes aos objetos mapeados em "models.py", caso elas não existam. A seguir, ele persiste os dados do repositório, ou seja, adiciona as informações do objeto "Repositório" no banco de dados. Por último, acontece o "commit", onde esses dados persistidos anteriormente são salvos no banco de dados. A seguir encontra-se o código do método "saveRepositoryOnDB(repository)":

```
def saveRepositoryOnDB(self, repository):

    # create a new session
    Session = sessionmaker(bind=self.engine)
    session = Session()

    # generate database schema
    declarative_base().metadata.create_all(self.engine)

```

```
    # persisting data
    session.add(repository)

    # commit and close session
    try:
        session.commit()
    except (sqlalchemy.exc.SQLAlchemyError, sqlalchemy.exc.DBAPIError)
    as e:
        print(e)
        session.rollback()
        raise

    session.close()
```

Após essa etapa, foram salvos 225 repositórios no banco de dados.

3.3.6 Critério de Seleção dos Projetos de Software

Mesmo com os dois filtros anteriores, um para buscar apenas repositórios com a *tag* BDD e um para retirar os repositórios que não possuem nenhum "feature", ainda é possível que alguns repositórios salvos não utilizem realmente BDD. Isso pode acontecer com repositórios que criaram a pasta "features" no início de seu desenvolvimento, mas ao longo dos dias, os arquivos de códigos foram sendo atualizados e a pasta "features" não. Os repositórios que caem nessa categoria são categorizados como falso positivos, pois aparentam ser dados válidos quando, na verdade, não são. Por essa razão, estes projetos não devem ser inclusos nesse estudo.

Para descobrir esses repositórios, foi utilizado um repositório OSS[3]. Este projeto adquire os projetos já salvos no banco de dados e calcula a diferença entre o último *update* da pasta "features" e o último *update* de algum código fonte, informações adquiridas através da API do *GitHub*. Essa diferença foi chamada de *delta* e, a partir dos *deltas* de todos os repositórios, foi criado o programa "discover_useless_repos.py", contido no repositório de mineração deste estudo. Esse programa é utilizado para descobrir esses repositórios que aparentam utilizar BDD, mas, na verdade, não utilizam. O programa abre um arquivo com os *deltas* dos repositórios minerados e cria um histograma usando a biblioteca "Matplotlib". O histograma pode ser visto na Figura 3.5, que possui, no eixo Y, o número de repositórios com certo *delta* e, no eixo X, o *delta* em si.

O código do programa "discover_useless_repos.py" pode ser visto a seguir:

```
import matplotlib.pyplot as plt

```

[3]https://github.com/RafaelFazzolino/test_pydriller

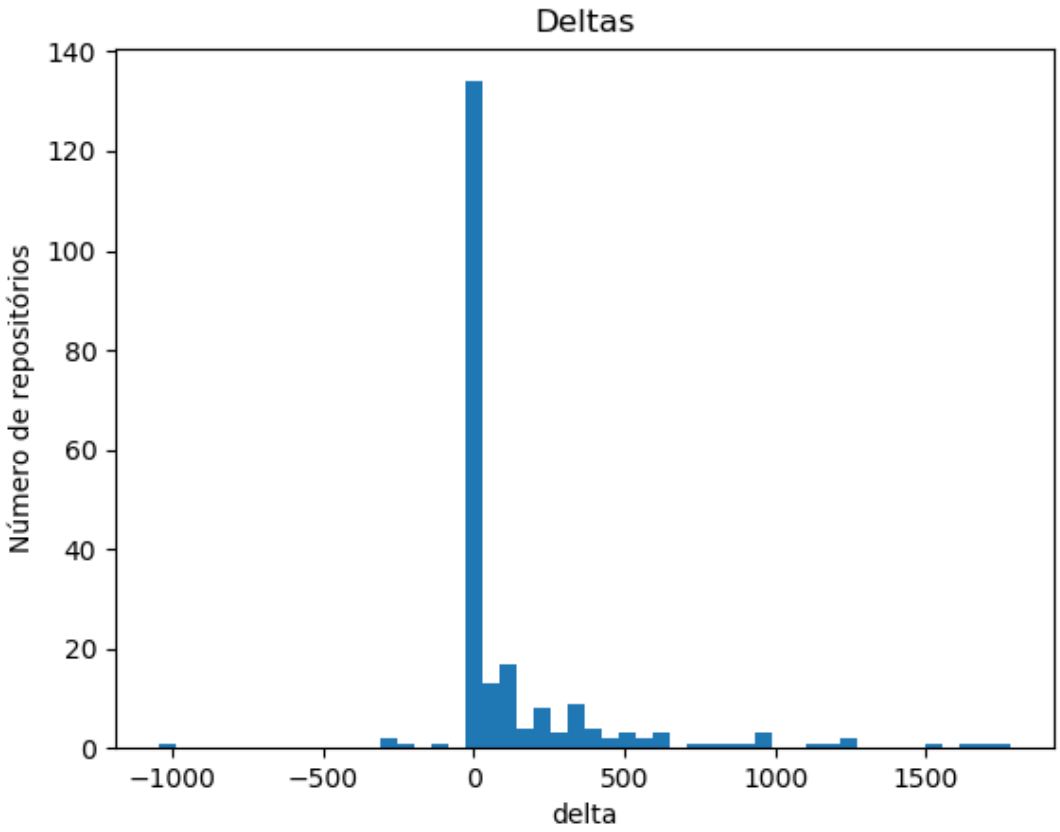


Figura 3.5: Histograma de comparação da última atualização do BDD com a última atualização do código fonte

```
f = open("result_delta.json", "r")
deltas = []
json = ast.literal_eval(f.read())

#creating graphic
plt.hist(deltas, bins=50)
plt.title("Deltas")
plt.ylabel("Numero de repositorios")
plt.xlabel("delta")
plt.show()
```

Com esses dados, foi decidido que os repositórios com *delta* maior que 350 seriam retirados do estudo, por serem considerados como falso positivos. O seguinte código, pertencente ao programa "discover_useless_repos.py", permitiu descobrir esses repositórios, que foram retirados manualmente:

```
for repo in json:
    deltas.append(repo['delta'])
    if repo['delta'] > 350:
```

```
        print(repo['project'])
```

No final, depois de retirados esses repositórios, remanesceram 186 repositórios, que foram considerados como válidos e utilizados para o estudo.

3.3.7 Compilação dos Dados para Análise

Para a compilação dos dados para análise, foi usado o programa "select_data.py". Para cada item necessário para análise, é invocado um método diferente do "view_model.py", que busca os dados necessários no banco de dados e retorna esses dados organizados para que o "select_data.py" possa construir seus gráficos. A seguir, temos um exemplo da parte do código do "select_data.py" que busca pelo número de cenários por *feature* nos 100 repositórios mais populares:

```
reposMostPopular = viewRepository.
    get100ReposMostPopular_ScenarioPerFeature()
print(reposMostPopular)
plt.boxplot(reposMostPopular['Number of Scenarios'])
plt.title("Number of Scenarios per Feature in 100 Most Popular
    Repositories")
plt.ylabel("Scenarios in one Feature")
plt.show()
```

O método "get100ReposMostPopular_ScenarioPerFeature()", por sua vez, conecta com o banco de dados, realiza uma *query* referente aos dados que busca no banco de dados, organiza os dados e retorna-os. A parte de conexão com o banco de dados pode ser visto no seguinte código:

```
# setting things
metadata = db.MetaData()
repositorios = db.Table('repository', metadata, autoload=True,
    autoload_with=self.engine)
features = db.Table('feature', metadata, autoload=True, autoload_with=
    self.engine)
scenarios = db.Table('scenario', metadata, autoload=True, autoload_with=
    self.engine)
Session = sessionmaker(bind=self.engine)
session = Session()
```

A parte de criação da *query* pode ser acompanhada a seguir:

```
# subquery dos 100 mais famosos
famososQuery = session.query(repositorios).order_by(repositorios.columns
    .stars.desc()).limit(100).subquery()

# query principal
```

```
sums = session.query(features.columns.idfeature.label("idfeature"), db.
    func.count(scenarios.columns.idscenario).label("Number of Scenarios")
    )

# JOIN
featureJoinScenario = features.join(scenarios, features.columns.
    idfeature == scenarios.columns.feature_id)
repositoryJoinFeatureJoinScenario = famososQuery.join(
    featureJoinScenario, famososQuery.columns.idrepository == features.
    columns.repository_id)
sums = sums.select_from(repositoryJoinFeatureJoinScenario)

# GROUP BY
sums = sums.group_by(features.columns.idfeature)

# SELECT
results = sums.all()
```

Finalmente, a parte de organização e retorno dos dados pode ser vista a seguir. Foi utilizada a biblioteca "Pandas" para a organização dos dados:

```
import pandas as pd

# Pandas organization
df = pd.DataFrame(results)
df.columns = results[0].keys()
df.head(5)
session.close()
return df
```

Essa mesma linha de raciocínio foi utilizada para para obtenção dos outros dados. Assim, existe um método para cada um dos seguintes itens:

- Número de *steps* por cenário nos 100 repositórios mais populares;
- Número de *steps* por cenário nos 100 repositórios mais recentes;
- Número de *steps* por cenário nos 100 repositórios com mais colaboradores;
- Número de *steps* por cenário nos repositórios que utilizam *Java*;
- Número de *steps* por cenário nos repositórios que utilizam *JavaScript*;
- Número de *steps* por cenário nos repositórios que utilizam *Python*;
- Número de *steps* por cenário nos repositórios que utilizam *Ruby*;
- Número de cenários por *feature* nos 100 repositórios mais populares;

- Número de cenários por *feature* nos 100 repositórios mais recentes;
- Número de cenários por *feature* nos 100 repositórios com mais colaboradores;
- Número de cenários por *feature* nos repositórios que utilizam *Java*;
- Número de cenários por *feature* nos repositórios que utilizam *JavaScript*;
- Número de cenários por *feature* nos repositórios que utilizam *Python*;
- Número de cenários por *feature* nos repositórios que utilizam *Ruby*;
- Número de repositórios que utiliza cada linguagem nos 100 repositórios mais populares;
- Número de repositórios que utiliza cada linguagem nos 100 repositórios mais recentes;
- Número de repositórios que utiliza cada linguagem nos 100 repositórios com mais colaboradores.

Os dados acima são usados para criar gráficos para análises, porém também é necessário a análise de certos dados usando o coeficiente de correlação de Pearson. Para esse tipo de análise, o método relativo do programa "select_data.py", ao invés de criar um gráfico com os dados retornados, calcula o coeficiente de Pearson utilizando a biblioteca "Scipy", assim como exemplificado no código abaixo:

```
1 from scipy.stats.stats import pearsonr
2
3 print("For 100 Most Popular Repositories:")
4 print("(Pearson's correlation coefficient, 2-tailed p-value) = ",
5       pearsonr(reposMostPopular["size"], reposMostPopular["Number of
     Features"]))
6 print()
```

O coeficiente de Pearson é calculado para os seguintes dados:

- Número de *features* do repositório por tamanho do repositório nos 100 repositórios mais populares;
- Número de *features* do repositório por tamanho do repositório nos 100 repositórios mais recentes;
- Número de *features* do repositório por tamanho do repositório nos 100 repositórios com mais colaboradores;
- Número de *features* do repositório por tamanho do repositório nos repositórios que utilizam *Java*;

- Número de *features* do repositório por tamanho do repositório nos repositórios que utilizam *JavaScript*;
- Número de *features* do repositório por tamanho do repositório nos repositórios que utilizam *Python*;
- Número de *features* do repositório por tamanho do repositório nos repositórios que utilizam *Ruby*.

Capítulo 4

Análise dos Resultados

4.1 Linguagens

Primeiramente, serão analisados os gráficos que dizem respeito ao número de linguagens de programação de projetos que usam BDD. São, no total, três gráficos, um referente aos 100 repositórios mais populares (Figura 4.1), um referente aos 100 repositórios com mais contribuidores (Figura 4.2) e um referente aos 100 repositórios que tiveram alguma atualização mais recente (Figura 4.3).

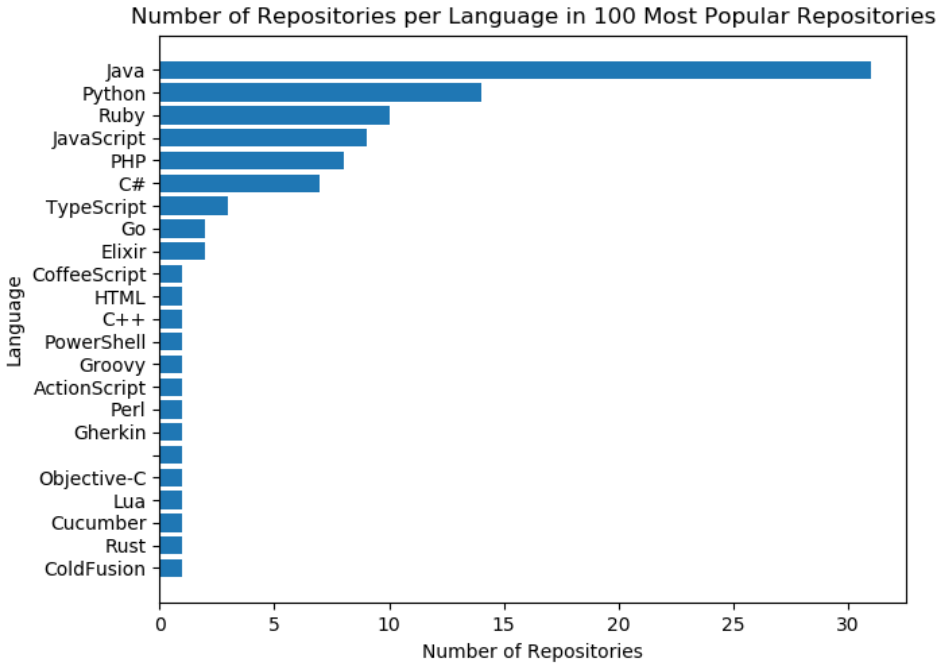


Figura 4.1: Número de repositórios por linguagem nos 100 repositórios mais populares

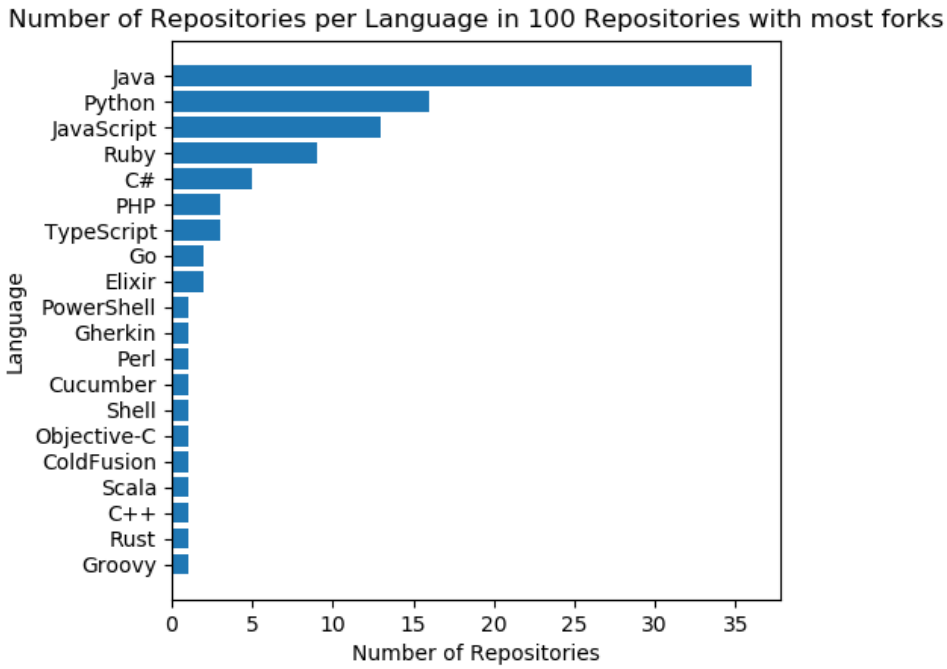


Figura 4.2: Número de repositórios por linguagem nos 100 repositórios com mais contribuidores

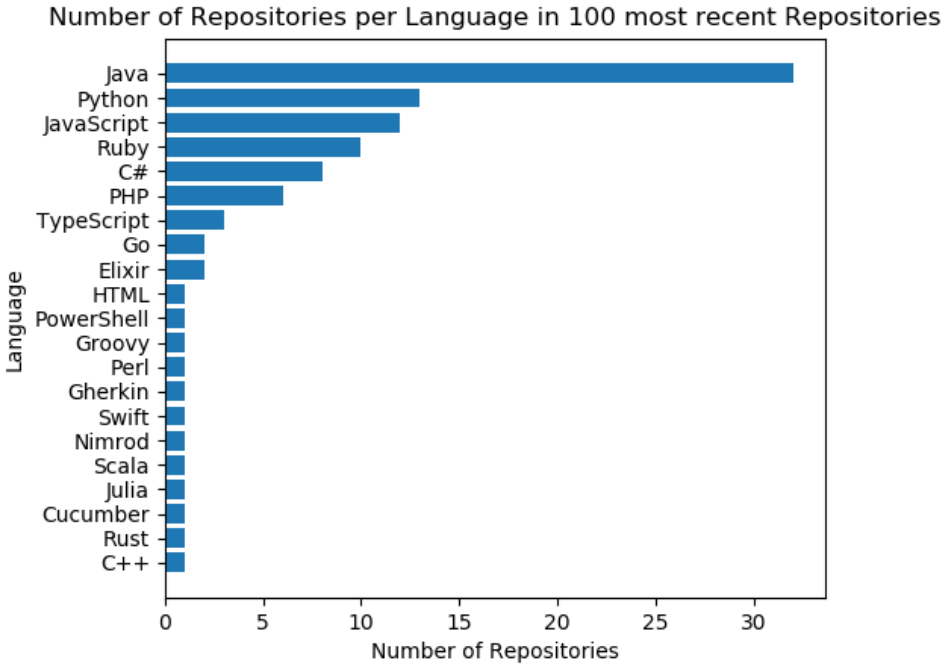


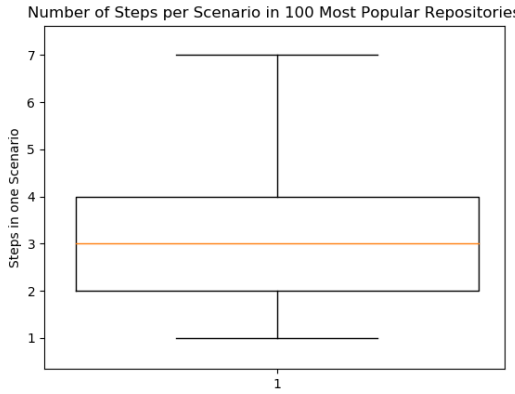
Figura 4.3: Número de repositórios por linguagem nos 100 repositórios mais recentes

Podemos perceber claramente que, em todos os gráficos, as quatro principais linguagens são *Java*, *Python*, *JavaScript* e *Ruby*, com *Java* liderando por uma grande margem.

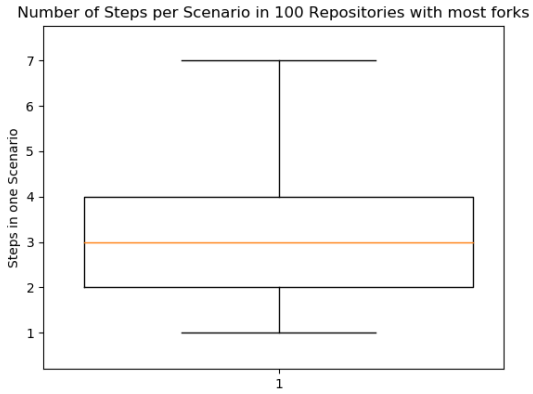
É possível ver também que as linguagens *C#* e *PHP* estão consistentemente atrás das quatro primeiras linguagens. Isso pode ser um indicativo de que estas duas linguagens estão ganhando seu espaço nesse meio.

4.2 Número de Steps por Cenários

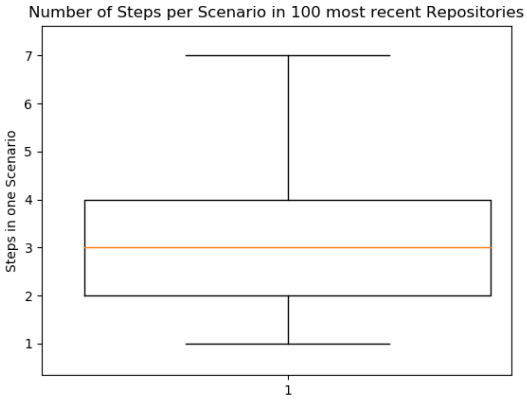
Serão analisados agora os gráficos referentes ao número de *steps* por cenário. Primeiro, serão comparados entre si os gráficos dos repositórios mais populares (Figura 4.4a), dos repositórios com mais contribuidores (Figura 4.4b) e dos repositórios mais recentes (Figura 4.4c).



(a) 100 repositórios mais populares



(b) 100 repositórios com mais contribuidores



(c) 100 repositórios mais recentes

Figura 4.4: Número de *steps* por cenário

Podemos ver que os três gráficos têm a mesma dispersão de dados, com o valor da mediana em 3, do mínimo em 1 e do máximo em 7, quartil inferior em 2 e quartil superior em 4. Todos os três possuem pontos discrepantes acima do 7 que foram desconsiderados por estarem muito longe de onde estão concentrados a maioria dos valores.

Assim, concluímos que a maioria dos projetos possuem de 2 a 4 *steps* em cada cenário, independente do ponto de vista que usamos. Em geral, a métrica recomendada é a de 3 a 8 *steps* por cenário [9]. Dessa forma, percebe-se que a comunidade tem seguido, em sua maioria, essa recomendação. No entanto, alguns questionamentos podem ser feitos, como o caso de cenários com apenas dois *steps*. Foi procurado manualmente nos arquivos de *feature* salvos localmente através do processo de mineração de dados deste estudo alguns desses casos em que o cenário possui apenas dois *steps* e a Figura 4.5 mostra um exemplo de arquivo nestes conformes.

```
47  @allow-rescue
48  Scenario: Attempt to access a resource I am not authorized to see
49    When I go to the last post's edit page
50    Then I should see "You are not authorized to perform this action"
51
52  Scenario: Viewing the default action items
53    When I follow "View"
54    Then I should not see an action item link to "Edit"
55
56  @allow-rescue
57  Scenario: Attempting to visit a Page without authorization
58    When I go to the admin no access page
59    Then I should see "You are not authorized to perform this action"
60
61  @allow-rescue
62  Scenario: Viewing a page with authorization
63    When I go to the admin dashboard page
64    Then I should see "Dashboard"
```

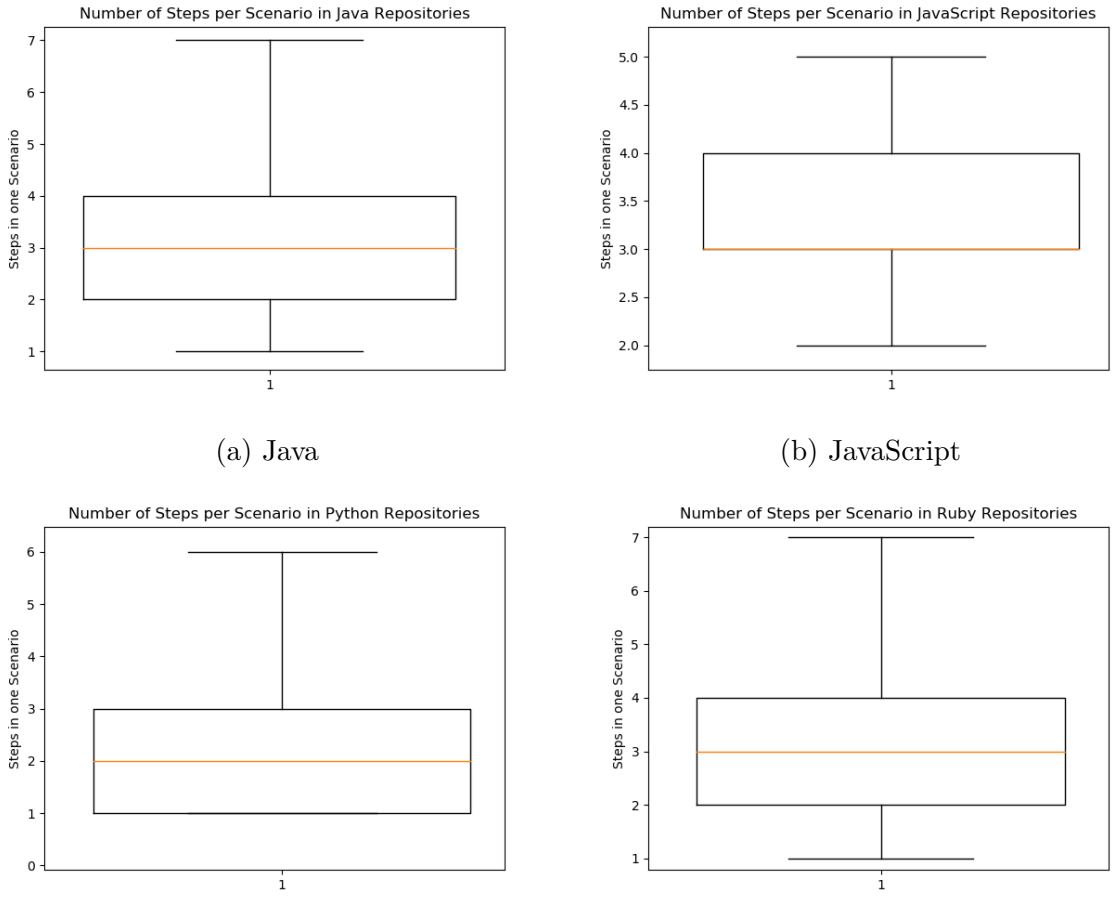
Figura 4.5: Exemplo de *feature* onde os cenários têm apenas 2 *steps*

Percebemos com este exemplo, que este cenário possui apenas dois *steps* e que utiliza apenas as palavras-chave *When* e *Then*. Segundo Smart [2], a ordem natural do cenário é *Given ... When ... Then*. As palavras chave *And* e *But* podem ser usadas para juntar vários *Given, When* ou *Then* de um jeito mais fácil de ler. Logo, estes cenários estão ignorando o step com a palavra-chave *Given*. Isso pode ser um problema, pois o *step* com a palavra-chave *Given* indica a condição do cenário, ou seja, o que é necessário acontecer anteriormente para que o cenário aconteça. Sem esse *step*, pode acontecer algum problema de comunicação e a *feature* pode acabar não atendendo às especificações necessárias.

Para entender a importância do *step* com a palavra-chave *Given*, verificaremos como exemplo o cenário "Attempting to visit a Page without authorization", presente na Figura

4.5. O objetivo dele é que uma pessoa sem autorização, quando tenta acessar a página de *admin*, receba uma mensagem de que ela não é autorizada a fazer essa ação. Porém, sem um *step* com a palavra-chave *Given* indicando de que o cenário só ocorre caso a pessoa não tenha autorização, o cenário pode ser interpretado como sendo válido para todos os casos, inclusive o que o usuário possui autorização.

A seguir, serão comparados os gráficos das quatro linguagens mais populares (Figura 4.6).



(a) Java

(b) JavaScript

(c) Python

(d) Ruby

Figura 4.6: Número de *steps* por cenário em repositórios das linguagens mais populares

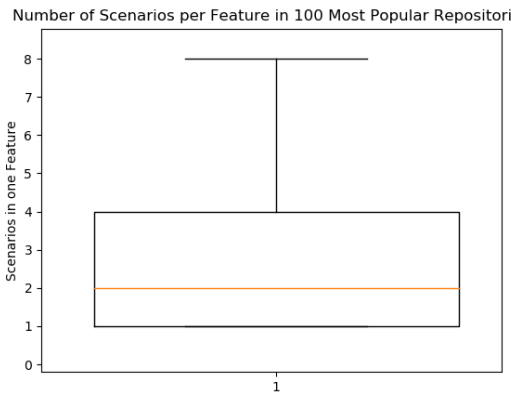
Podemos notar que os números dos gráficos de *Java* e de *Ruby* possuem dispersão igual às dos gráficos dos 100 repositórios mais populares, dos 100 repositórios com mais contribuidores e dos 100 mais recentes. Portanto, as mesmas conclusões podem ser retiradas sobre eles. Os gráficos de *JavaScript* e *Python*, porém, possuem dispersões diferentes.

O gráfico de *JavaScript* possui números mais concentrados e mais altos, com valores mínimo e máximo sendo 2 e 5, respectivamente, e quartis inferior e superior sendo 3 e 4, respectivamente. Esses dados revelam que a maioria dos cenários tem 3 ou 4 *steps* e,portanto, que os projetos que utilizam *JavaScript* estão mais propensos a seguir a recomendação de 3 a 8 *steps* por cenário [9].

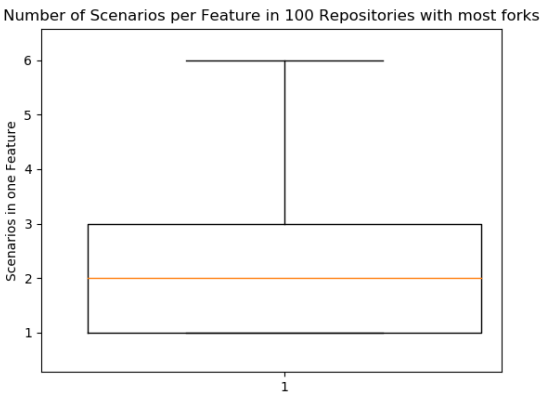
Já o gráfico de *Python* possui valores mais baixos, com concentração maior. Sua média de *steps* por cenário é de 1 a 3, valor dos quartis inferior e superior, respectivamente. Analisando a mediana desse gráfico, vemos que ela é 2. Isso significa que, se ordenarmos todos os valores em ordem crescente, o valor exatamente no meio é 2. Podemos tirar uma conclusão disso de que, no mínimo, metade dos repositórios que utilizam a linguagem *Python* tem 2 ou menos *steps* por cenário. Esses números são preocupantes por mostrarem que há vários repositórios com o número de *steps* por cenário abaixo do mínimo recomendado, que é 3 [9].

4.3 Número de Cenários por Feature

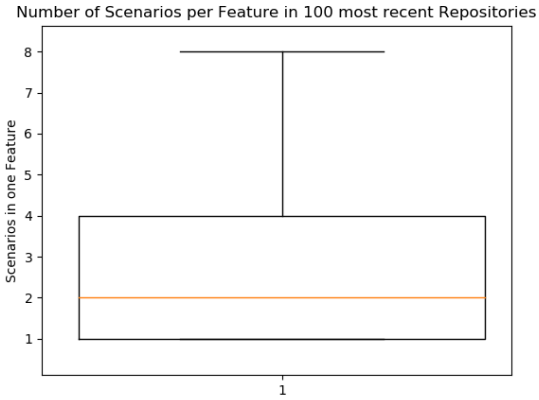
Assim como a seção anterior, começaremos analisando entre si os gráficos dos repositórios mais populares (Figura 4.7a), dos repositórios com mais contribuidores (Figura 4.7b) e dos repositórios mais recentes (Figura 4.7c).



(a) 100 repositórios mais populares



(b) 100 repositórios com mais contribuidores

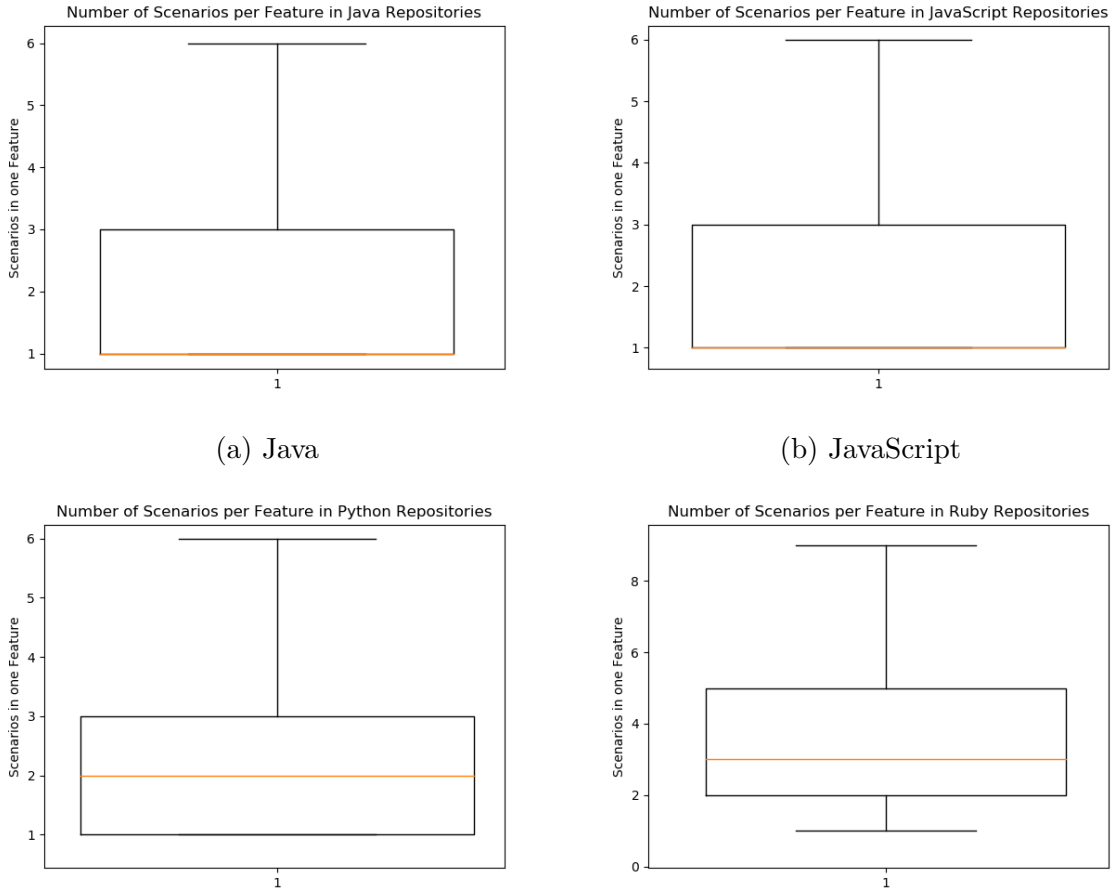


(c) 100 repositórios mais recentes

Figura 4.7: Número de cenários por *feature*

O primeiro ponto que percebemos é que os três gráficos apresentam uma dispersão bastante similar entre si. Todos os três têm limite mínimo 1, quartil inferior 1 e mediana 2. As diferenças estão no quartil superior e no limite máximo. Enquanto que o quartil superior dos repositórios com mais colaboradores é 3 e o limite máximo é 6, os dois outros gráficos possuem quartil superior 4 e limite 8. A conclusão que tiramos disso é que os repositórios tendem a ter de 1 a 4 cenários por *feature* no geral, mas quando os projetos tendem a ter mais colaboradores, a tendência situa-se entre 1 a 3 cenários por *feature*.

Agora serão comparados os gráficos de número de cenários por *feature* dos repositórios de cada uma das linguagens de programação mais usadas (Figura 4.8).



(a) Java

(b) JavaScript

(c) Python

(d) Ruby

Figura 4.8: Número de cenários por *feature* em repositórios das linguagens mais populares

Analisando os gráficos, vemos que os repositórios de linguagens *Java*, *JavaScript* e *Ruby* todos possuem, em média, de 1 a 3 cenários em cada *feature*, enquanto que os repositórios de linguagem *Ruby* possuem uma média de 2 a 5 cenários por *feature*. A média para os repositórios que utilizam *Ruby* provavelmente se mostrou mais alta por causa da natureza da linguagem de programação.

Um ponto interessante sobre os gráficos de *Java* e *JavaScript* é que ambos possuem quartil inferior, mediana e limite inferior todos com o mesmo valor 1. Isso implica que pelo menos metade dos valores são 1, ou seja, para repositórios que usam *Java* e *JavaScript* como linguagem, no mínimo metade deles possuem apenas um cenário por *feature*.

Como foram resultados bem distintos, considerando os mínimos e máximos, chegamos à conclusão de que os repositórios tendem a ter, em geral, de 1 a 5 cenários por *feature*, com variações nesse intervalo dependendo do ponto de vista observado.

Segundo North [9], não há regras para o número de cenários ideal, porém se um *feature* possui cinco ou seis cenários, algum cenário provavelmente pode ser agrupado com outro cenário parecido. Isso nos leva a crer que o nosso resultado de 1 a 5 cenários por *feature* é um ótimo resultado e atende à proposição da ferramenta, ou seja, os repositórios estão usando o número de cenários por *feature* esperado pelo criador do BDD.

4.4 Número de Features por Tamanho de Repositório

Calculando o coeficiente de correlação de Pearson entre o tamanho do repositório, em Kilobytes, e seu número de features, obtivemos estes resultados:

- 100 repositórios mais populares: 0.008378587217726694
- 100 repositórios com mais contribuidores: -0.00020566898942679483
- 100 repositórios mais recentes: 0.0069092705164275695

Como podemos perceber analisando o coeficiente de Pearson, essas duas variáveis muito provavelmente não possuem correlação, pois os valores dos coeficientes foram muito próximos de zero.

Os coeficientes de Pearson calculados para as quatro linguagens de programação mais populares foram:

- *Java*: 0.024504317215782376
- *JavaScript*: 0.028927433099483216
- *Python*: -0.044335215795885646
- *Ruby*: -0.06176734886824657

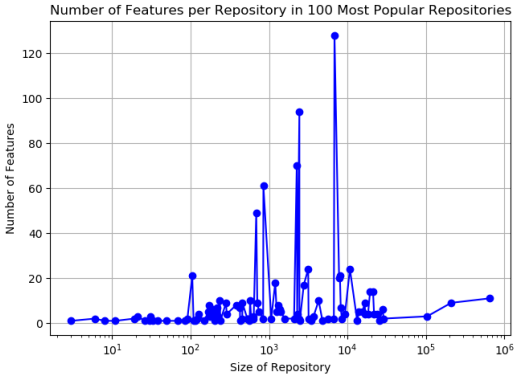
Os coeficientes deram um pouco mais altos nessa análise, porém ainda estão muito próximos de zero. Esses valores um pouco mais elevados em relação aos outros valores devem-se, principalmente, pelo número menor de repositórios em cada linguagem, pois é uma amostra bem menor.

Para Cohen [26], valores entre 0,10 e 0,29 podem ser considerados pequenos, valores entre 0,30 e 0,49 podem ser interpretados como médios; e valores entre 0,50 e 1 podem ser entendidos como grandes. Dancey e Reidy [27] classificam de maneira diferente: r = 0,10 até 0,30 (fraco); r = 0,40 até 0,6 (moderado); r = 0,70 até 1 (forte).

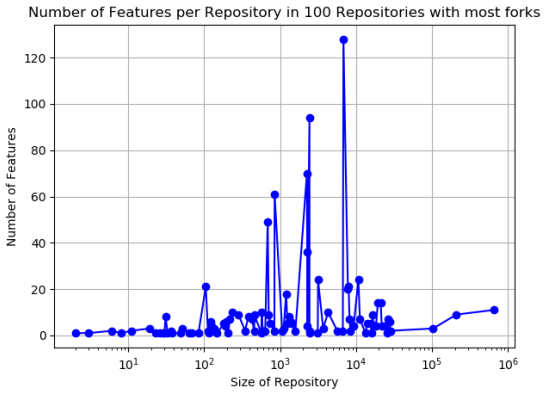
Com base nisso e nos dados obtidos, é possível concluir, então, que não há relação direta entre o tamanho do repositório e o seu número de *features*, pois todos os valores calculados são menores do que o menor intervalo que pode ser considerado pelos dois autores.

Para essa análise, também foram feitos gráficos de linha para termos uma visualização geral de como os dados estão dispostos, porém o fator definitivo para validade desse ponto é o coeficiente de correlação de Pearson. É importante notar que o eixo do número de repositórios está na escala logarítmica. Isso foi feito apenas para melhor visualização dos dados.

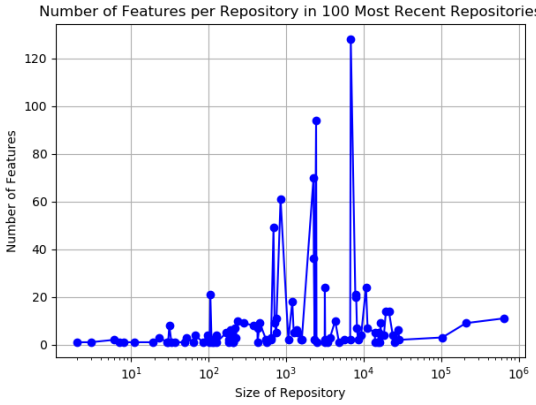
A seguir temos os gráficos dos repositórios mais populares (Figura 4.9a), dos repositórios com mais contribuidores (Figura 4.9b)e dos repositórios mais recentes (Figura 4.9c).



(a) 100 repositórios mais populares



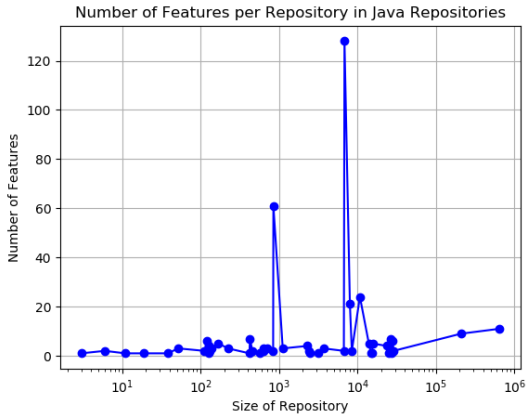
(b) 100 repositórios com mais contribuidores



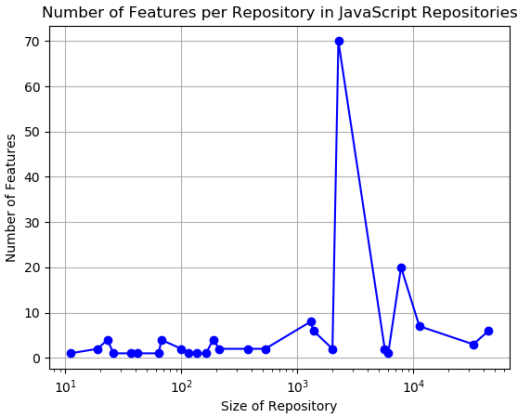
(c) 100 repositórios mais recentes

Figura 4.9: Número de Features por tamanho do repositório

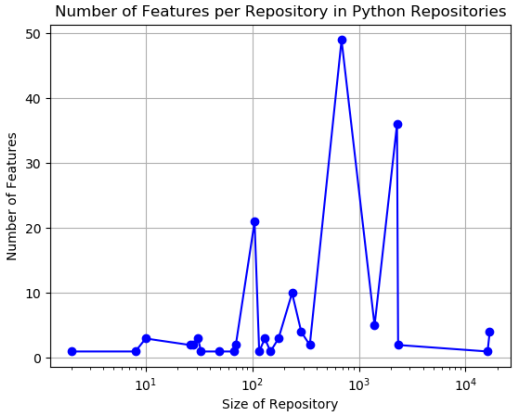
Finalmente, temos os gráficos referentes às linguagens de programação mais populares (Figura 4.10).



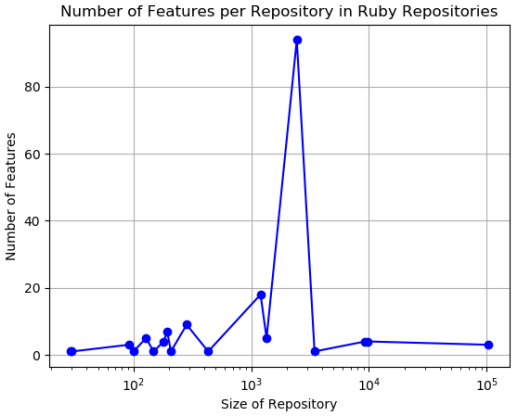
(a) Java



(b) JavaScript



(c) Python



(d) Ruby

Figura 4.10: Número de Features por tamanho do repositório em repositórios das linguagens mais populares

Capítulo 5

Conclusões

Com este estudo de caso, foi possível aprender mais sobre projetos no *GitHub* que utilizam Behavior Driven Development.

Foi observado que as quatro linguagens mais utilizadas por repositórios que usam BDD são *Java*, *JavaScript*, *Python* e *Ruby*, com *Java* estando em primeiro lugar por uma grande margem. Esse resultado se manteve constante mesmo usando diferentes pontos de vista. Os diferentes pontos de vista usados foram os repositórios mais populares, os repositórios mais recentes e os repositórios com mais contribuidores.

Quanto ao número médio de *steps* por cenário, foi constatado que os repositórios possuem de 2 a 4 *steps* por cenário. Como a métrica recomendada é de 3 a 8 *steps* por cenário, concluiu-se que parte dos desenvolvedores estão seguindo essa métrica. Porém, existem projetos que utilizam menos de 3 *steps* em seus cenários, o que não é o ideal, pois pode levar a problemas de comunicação.

Também foi constatado que os repositórios que utilizam BDD tendem a ter de 1 a 5 cenários por *feature*, com pequenas variações nesse intervalo dependendo do ponto de vista observado. Não há regras para o número de cenários ideal, porém se um *feature* possui cinco ou seis cenários, algum cenário provavelmente pode ser agrupado com outro cenário parecido. Isso nos leva a crer que o nosso resultado de 1 a 5 cenários por *feature* atende à proposição da ferramenta, ou seja, os repositórios estão usando o número de cenários por *feature* recomendado.

Sobre a possibilidade de relação entre o tamanho do repositório e o seu número de *features*, ela foi descartada. Isso aconteceu porque foi calculado, sobre vários pontos de vista, o coeficiente correlação de Pearson, que indica se duas variáveis possuem correlação entre si e o coeficiente indicou, em todos os pontos de vista, que o tamanho do repositório não possui relação com o seu número de *features*.

Quanto aos instrumentos utilizados, ou seja, a API do *GitHub*, a linguagem utilizada, as bibliotecas e o banco de dados, foi concluído que eles foram satisfatórios. Não houve

nenhum problema durante a utilização deles. Pelo contrário, eles facilitaram o desenvolvimento do projeto.

Para o futuro, mais informações podem ser coletadas utilizando outro método de pesquisa, uma vez que este estudo procurou armazenar mais dados do que pretendia utilizar. Um exemplo de pesquisa útil seria um questionário direcionado aos donos dos projetos que usam BDD, cujos *e-mails* foram salvos durante essa pesquisa.

Referências

[1] *Test Driven Development is the best thing that has happened to software design.* https://www.thoughtworks.com/insights/blog/test-driven-development-best-thing-has-happened-software-design, Accessed: August 24, 2019. ix, 5

[2] Smart, John Ferguson: *Bdd in action: behavior-driven development for the whole software lifecycle*. 2015. ix, 1, 4, 5, 6, 7, 34

[3] Nair, Jithin: *TDD vs BDD – What's the Difference Between TDD and BDD?*, April 18th, 2018 (accessed August 22, 2019). https://blog.testlodge.com/tdd-vs-bdd/. ix, 6, 7

[4] Fayyad, Usama, Gregory Piatetsky-Shapiro e Padhraic Smyth: *From data mining to knowledge discovery in databases*. AI magazine, 17(3):37–37, 1996. ix, 8, 9

[5] *File:Elements of a boxplot.svg.* https://commons.wikimedia.org/wiki/File:Elements_of_a_boxplot.svg, Accessed: August 24, 2019. ix, 15

[6] Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries *et al.*: *Manifesto for agile software development*. 2001. 1

[7] Boehm, Barry: *Get ready for agile methods, with care*. Computer, (1):64–69, 2002. 1

[8] Beck, Kent: *Test-driven development: by example*. Addison-Wesley Professional, 2003. 1, 4

[9] North, Dan: *Introducing BDD*, 2006 (accessed June 20, 2019). https://dannorth.net/introducing-bdd/. 1, 4, 5, 34, 36, 39

[10] Evans, Eric: *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004. 4, 5

[11] Janzen, David e Hossein Saiedian: *Test-driven development concepts, taxonomy, and future direction*. Computer, 38(9):43–50, 2005. 4

[12] Group, Standish *et al.*: *Chaos report.(2002)*. West Yarmouth (MA): The Standish Group, 2008. 5

[13] Hellesøy, Aslak: *Introduction to Cucumber*, 2019 (accessed June 20, 2019). https://cucumber.io/docs/guides/overview/. 5, 6

[14] Vieira, Juracy e Nicholas Pufal: *3 misconceptions about BDD*, 2013 (accessed July 01, 2019). https://www.thoughtworks.com/insights/blog/3-misconceptions-about-bdd. 7

[15] Agilealliance: *BDD: Learn about Behavior Driven Development | Agile Alliance*, (accessed July 01, 2019). https://www.agilealliance.org/glossary/bdd/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'bdd))~searchTerm~'~sort~false~sortDirection~'asc~page~1). 7

[16] Hand, David, Heikki Mannila e Padhraic Smyth: *Principles of data mining*. 2001. 8

[17] Hand, David J: *Data mining*. Encyclopedia of Environmetrics, 2, 2006. 8, 9

[18] Goldschmidt, Ronaldo e Emmanuel Passos: *Data mining: um guia prático*. Gulf Professional Publishing, 2005. 8

[19] Kalliamvakou, Eirini, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German e Daniela Damian: *The promises and perils of mining github*. Em *Proceedings of the 11th working conference on mining software repositories*, páginas 92–101. ACM, 2014. 9, 10

[20] GitHub: *GitHub Documentation*, 2019 (accessed August 22, 2019). https://developer.github.com/v3/. 9

[21] Zelkowitz, Marvin V e Dolores R. Wallace: *Experimental models for validating technology*. Computer, 31(5):23–31, 1998. 11

[22] Creswell, John W: *Research design*. Qualitative and Quantitative Approaches, 1994. 11

[23] Wohlin, Claes, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell e Anders Wesslén: *Experimentation in software engineering*. Springer Science & Business Media, 2012. 11, 12, 13, 16

[24] Briand, Lionel C, Christiane M Differding e H Dieter Rombach: *Practical guidelines for measurement-based process improvement*. Software Process: Improvement and Practice, 2(4):253–280, 1996. 11, 12

[25] Yin, Robert K: *Case Study Research and Applications: Design and Methods*. SAGE Publications, 2017. 11

[26] Cohen, J: *Statistical power analysis for the behavioral sciences . hillsdale, nj: Eribaum*, 1988. 40

[27] Dancey, Christine e John Reidy: *Estatística Sem Matemática para Psicologia-7*. Penso Editora, 2018. 40

Anexo I

Repositórios

Repositórios usados neste projeto:

https://api.github.com/repos/karma-runner/karma
https://api.github.com/repos/pester/Pester
https://api.github.com/repos/cucumber/aruba
https://api.github.com/repos/codegram/spinach
https://api.github.com/repos/DATA-DOG/godog
https://api.github.com/repos/continuumsecurity/bdd-security
https://api.github.com/repos/acuminous/yadda
https://api.github.com/repos/rlisagor/freshen
https://api.github.com/repos/jan-molak/serenity-js
https://api.github.com/repos/meadsteve/white-bread
https://api.github.com/repos/salad/salad
https://api.github.com/repos/cpp-testing/GUnit
https://api.github.com/repos/calabash/x-platform-example
https://api.github.com/repos/paulgibbs/behat-wordpress-extension
https://api.github.com/repos/behave/behave-django
https://api.github.com/repos/BitBagCommerce/SyliusCmsPlugin
https://api.github.com/repos/gauravkarvir/cucumber_testng_java
https://api.github.com/repos/qmetry/qaf
https://api.github.com/repos/serenity-bdd/serenity-cucumber
https://api.github.com/repos/machzqcq/CucumberJVMExamples
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java
https://api.github.com/repos/Halleck45/BDDWizard
https://api.github.com/repos/cabbage-ex/cabbage
https://api.github.com/repos/everzet/fullstack-bdd-sflive2012
https://api.github.com/repos/pjlsergeant/test-bdd-cucumber-perl
https://api.github.com/repos/noblesamurai/cucumis

https://api.github.com/repos/hidroh/cucumber-api
https://api.github.com/repos/ekino/veggies
https://api.github.com/repos/tobiasheine/Movies
https://api.github.com/repos/matt-allan/quicksilver
https://api.github.com/repos/Chorus-bdd/Chorus
https://api.github.com/repos/fesor/json_spec
https://api.github.com/repos/mikek/behave-http
https://api.github.com/repos/angiejones/restassured-with-cucumber-demo
https://api.github.com/repos/dcypherthis/wdio-boilerplate-cucumber
https://api.github.com/repos/ustwo/bdd-crossplatform-apps
https://api.github.com/repos/flashquartermaster/Cuke4AS3
https://api.github.com/repos/priyankshah217/AppiumCucumberTest
https://api.github.com/repos/serenity-bdd/serenity-documentation
https://api.github.com/repos/yoshuawuyts/cuke-tap
https://api.github.com/repos/PredixDev/predix-bdd-cucumber
https://api.github.com/repos/ttutisani/Xunit.Gherkin.Quick
https://api.github.com/repos/Shashikant86/cucumber-appium
https://api.github.com/repos/graze/cucumber-rest-bdd
https://api.github.com/repos/CodemateLtd/Android-Cucumber-BDD-Sample
https://api.github.com/repos/bmsantos/cola-tests
https://api.github.com/repos/tychobrailleur/cucumber-jvm-examples
https://api.github.com/repos/bbqsrc/cucumber-rust
https://api.github.com/repos/ravolt/BravoDelta
https://api.github.com/repos/pib/calabash
https://api.github.com/repos/Fidano/ColdCukes
https://api.github.com/repos/sorin-costea/bdd
https://api.github.com/repos/stanfy/behave-rest
https://api.github.com/repos/mvallebr/BDD-java-cucumber-selenium
https://api.github.com/repos/gabrielfalcao/tornado-bdd-sample
https://api.github.com/repos/angiejones/automation-framework
https://api.github.com/repos/Galad/tranquire
https://api.github.com/repos/DavidSouther/tdd-angular
https://api.github.com/repos/rpfujiw/gui-auto-uat
https://api.github.com/repos/RobGibbens/BddWithXamarinUITest
https://api.github.com/repos/serenity-bdd/serenity-cucumber-starter
https://api.github.com/repos/speciphy/speciphy
https://api.github.com/repos/havvg/BDD-Experiment
https://api.github.com/repos/serhatbolsu/appium-python-bdd
https://api.github.com/repos/larryg01/klassi-js
https://api.github.com/repos/coshx/docker-bdd

https://api.github.com/repos/balamaci/blog-ui-bdd-testing
https://api.github.com/repos/oleg-toporkov/python-bdd-selenium
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java-maven-example
https://api.github.com/repos/giggio-samples/Exemplo-BDD-com-CSharp-Specflow-e-Headless
    -Browser
https://api.github.com/repos/thiagodp/concordialang
https://api.github.com/repos/Remchi/libapp
https://api.github.com/repos/mikehale/chef-bdd
https://api.github.com/repos/jharmn/TwitterRestTests
https://api.github.com/repos/sizuhiko/cakephp3-bdd-example
https://api.github.com/repos/sbtqa/api-factory
https://api.github.com/repos/johnnonolan/ChessBoard-Kata
https://api.github.com/repos/rahulrathore44/SeleniumCucumber
https://api.github.com/repos/20tab/django-bdd-toolkit
https://api.github.com/repos/harlanji/gogherkit
https://api.github.com/repos/tinyerp/openobject-mirliton
https://api.github.com/repos/marcusoftnet/BDDWithSpecFlow
https://api.github.com/repos/hdorgeval/cucumber-ts-starter
https://api.github.com/repos/olegpidsadnyi/pytest-bdd-example
https://api.github.com/repos/pradeepta02/python-behave-automation-framework
https://api.github.com/repos/shankybnl/selenium_BDD_framework
https://api.github.com/repos/jamesotron/Baseline
https://api.github.com/repos/petersuhm/laravel-bdd
https://api.github.com/repos/joshski/cucumbers-on-vine-hill
https://api.github.com/repos/chwilliamson/Vizregress
https://api.github.com/repos/serenity-bdd/bdd-trader
https://api.github.com/repos/ashenwgt/awesome-bdd-with-cucumber
https://api.github.com/repos/AndyLPK247/behavior-driven-python
https://api.github.com/repos/testdrivenio/django-aloe-bdd
https://api.github.com/repos/jharmn/api-bdd-tests
https://api.github.com/repos/readyapi/testserver-cucumber
https://api.github.com/repos/sauliuz/nightmare-cucumberjs
https://api.github.com/repos/mattwynne/bdd-as-if-you-meant-it
https://api.github.com/repos/cliftonc/drupal_behat
https://api.github.com/repos/colinbut/sales-order-system
https://api.github.com/repos/malavbhavsar/BestBay
https://api.github.com/repos/louisandkatherine/ti-uiautomator
https://api.github.com/repos/dajudge/testee.fi
https://api.github.com/repos/fsouza/lettuce_hudson_fabric
https://api.github.com/repos/daniel-hall/TestKit

https://api.github.com/repos/katalon-studio-samples/katalon-bdd-cucumber-tests
https://api.github.com/repos/spilio/apispots-bdd
https://api.github.com/repos/ing-bank/bdd-mobile-security-automation-framework
https://api.github.com/repos/Qualitestuk/SerenityBDD-Appium-Cross-Platform-Testing
    -Solution
https://api.github.com/repos/skalnik/TDD-BDD-Example
https://api.github.com/repos/rafaelcruz-net/BDD
https://api.github.com/repos/giggio-samples/Exemplo-Blog-BDD-CSharp
https://api.github.com/repos/mookid8000/DillPickle
https://api.github.com/repos/fdiasdev/bdd-behat
https://api.github.com/repos/hammernight/etsy_cucumber_taza_tutorial
https://api.github.com/repos/jagilpe/bdd-pact-microservices
https://api.github.com/repos/mattwynne/bdd-exchange-london-2011-code
https://api.github.com/repos/erikedin/ExecutableSpecifications.jl
https://api.github.com/repos/giggio-samples/outro-exemplo-de-bdd-com-csharp
https://api.github.com/repos/shaunc/cucumber_nim
https://api.github.com/repos/Chorus-bdd/chorus-js
https://api.github.com/repos/spyoungtech/behave-webdriver
https://api.github.com/repos/Frameworkium/frameworkium-bdd
https://api.github.com/repos/blu-corner/etiqet
https://api.github.com/repos/Rhoynar/python-selenium-bdd
https://api.github.com/repos/mhurwi/cucumber-meteor-tutorial
https://api.github.com/repos/alexgarzao/victory.backup
https://api.github.com/repos/meza/AAO
https://api.github.com/repos/serenity-js/tutorial-from-scripts-to-serenity
https://api.github.com/repos/sukesh15/BDDWorkshop
https://api.github.com/repos/phodal/bdd-frameworks-compare
https://api.github.com/repos/dunossauro/python-bdd-compare
https://api.github.com/repos/vlasar/bdd_on_rails4_2
https://api.github.com/repos/cedaniel200/base_project_for_web_automation_projects
https://api.github.com/repos/navalkgupta/protractor-cucumber-tests
https://api.github.com/repos/anair-it/drools-cucumber
https://api.github.com/repos/llonchj/browsersteps
https://api.github.com/repos/martinschneider/justtestlah
https://api.github.com/repos/Xceptance/neodymium-library
https://api.github.com/repos/lucascourot/PHPKataTrainReservation
https://api.github.com/repos/wesovilabs/gherkinize
https://api.github.com/repos/dariusz-wozniak/TddBook-Code
https://api.github.com/repos/serenity-bdd/serenity-exercises
https://api.github.com/repos/sabre1041/ose-bdd-demo

https://api.github.com/repos/jholsgrove/BDDRestSharp
https://api.github.com/repos/DanteLore/bdd-spark
https://api.github.com/repos/mica16/BDD-TDD-Demo
https://api.github.com/repos/devpaul/node-bdd-example
https://api.github.com/repos/jenkins-x/godog-jx
https://api.github.com/repos/MarkBorcherding/AgentBdd
https://api.github.com/repos/raghuk9/PhotonLabs_AutomationFramework_BDD_TDMS_MDD
https://api.github.com/repos/the-creative-tester/python-behave-web-browser-automation
    -example
https://api.github.com/repos/joshuaswarren/bdd
https://api.github.com/repos/behave-restful/behave-restful
https://api.github.com/repos/blaix/bddgen
https://api.github.com/repos/serenity-js/seed-cucumber
https://api.github.com/repos/kinisoftware/BDD-Java
https://api.github.com/repos/BitBagCommerce/SyliusBDD
https://api.github.com/repos/jbandi/bowling-bdd-java
https://api.github.com/repos/exu/php-bdd-bootstrap
https://api.github.com/repos/dwilkie/rails_bdd_course
https://api.github.com/repos/waheedahmed55/Hybrid-Selenium-Cucumber-BDD
https://api.github.com/repos/matason/behat-drupal
https://api.github.com/repos/inv/t4
https://api.github.com/repos/Nethereum/Nethereum.TestRPCRunner
https://api.github.com/repos/AlexandrosD/ng2-protractor-cucumber
https://api.github.com/repos/froots/snippetron
https://api.github.com/repos/joymax/lettuce-web
https://api.github.com/repos/haschi/dominium
https://api.github.com/repos/katalisha/behat-tutorial
https://api.github.com/repos/davidemoro/cookiecutter-qa
https://api.github.com/repos/ericyahhh/spectron-cucumber-example
https://api.github.com/repos/symonk/cucumber-jvm-selenium-allure2-test-automation
    -framework
https://api.github.com/repos/harry-kalligeros/serenityjs-demo
https://api.github.com/repos/ukrvassabi/serenity-bdd-test
https://api.github.com/repos/julialiuliu/Appium-BDD-python-mobile-app-testing
https://api.github.com/repos/serenity-bdd/serenity-cli
https://api.github.com/repos/js-republic/cucumber-slice-by-slice-workshop
https://api.github.com/repos/stmllr/TYPO3-BDD
https://api.github.com/repos/Mah-D/DwarfCucumber
https://api.github.com/repos/haroon-sheikh/cucumber-appium-ruby-example
https://api.github.com/repos/serenity-dojo/caffeinate-me

https://api.github.com/repos/marcusoftnet/BDDAsYouMeanIt
https://api.github.com/repos/sizuhiko/BddExampleApp
https://api.github.com/repos/sitture/cucumber-jvm-extentreport
https://api.github.com/repos/yurireeis/bddocs

100 repositórios mais populares:

https://api.github.com/repos/karma-runner/karma
https://api.github.com/repos/pester/Pester
https://api.github.com/repos/cucumber/aruba
https://api.github.com/repos/DATA-DOG/godog
https://api.github.com/repos/codegram/spinach
https://api.github.com/repos/continuumsecurity/bdd-security
https://api.github.com/repos/acuminous/yadda
https://api.github.com/repos/rlisagor/freshen
https://api.github.com/repos/jan-molak/serenity-js
https://api.github.com/repos/meadsteve/white-bread
https://api.github.com/repos/salad/salad
https://api.github.com/repos/BitBagCommerce/SyliusCmsPlugin
https://api.github.com/repos/cpp-testing/GUnit
https://api.github.com/repos/paulgibbs/behat-wordpress-extension
https://api.github.com/repos/behave/behave-django
https://api.github.com/repos/calabash/x-platform-example
https://api.github.com/repos/qmetry/qaf
https://api.github.com/repos/gauravkarvir/cucumber_testng_java
https://api.github.com/repos/serenity-bdd/serenity-cucumber
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java
https://api.github.com/repos/machzqcq/CucumberJVMExamples
https://api.github.com/repos/cabbage-ex/cabbage
https://api.github.com/repos/Halleck45/BDDWizard
https://api.github.com/repos/ekino/veggies
https://api.github.com/repos/everzet/fullstack-bdd-sflive2012
https://api.github.com/repos/hidroh/cucumber-api
https://api.github.com/repos/pjlsergeant/test-bdd-cucumber-perl
https://api.github.com/repos/noblesamurai/cucumis
https://api.github.com/repos/ttutisani/Xunit.Gherkin.Quick
https://api.github.com/repos/Xceptance/neodymium-library
https://api.github.com/repos/bbqsrc/cucumber-rust
https://api.github.com/repos/angiejones/restassured-with-cucumber-demo
https://api.github.com/repos/tobiasheine/Movies

https://api.github.com/repos/matt-allan/quicksilver
https://api.github.com/repos/mikek/behave-http
https://api.github.com/repos/Chorus-bdd/Chorus
https://api.github.com/repos/fesor/json_spec
https://api.github.com/repos/serenity-bdd/serenity-cucumber-starter
https://api.github.com/repos/ustwo/bdd-crossplatform-apps
https://api.github.com/repos/dcypherthis/wdio-boilerplate-cucumber
https://api.github.com/repos/flashquartermaster/Cuke4AS3
https://api.github.com/repos/priyankshah217/AppiumCucumberTest
https://api.github.com/repos/serenity-bdd/serenity-documentation
https://api.github.com/repos/graze/cucumber-rest-bdd
https://api.github.com/repos/yoshuawuyts/cuke-tap
https://api.github.com/repos/CodemateLtd/Android-Cucumber-BDD-Sample
https://api.github.com/repos/PredixDev/predix-bdd-cucumber
https://api.github.com/repos/angiejones/automation-framework
https://api.github.com/repos/Shashikant86/cucumber-appium
https://api.github.com/repos/ing-bank/bdd-mobile-security-automation-framework
https://api.github.com/repos/bmsantos/cola-tests
https://api.github.com/repos/Galad/tranquire
https://api.github.com/repos/larryg01/klassi-js
https://api.github.com/repos/hdorgeval/cucumber-ts-starter
https://api.github.com/repos/ravolt/BravoDelta
https://api.github.com/repos/AndyLPK247/behavior-driven-python
https://api.github.com/repos/tychobrailleur/cucumber-jvm-examples
https://api.github.com/repos/pib/calabash
https://api.github.com/repos/sorin-costea/bdd
https://api.github.com/repos/thiagodp/concordialang
https://api.github.com/repos/Rhoynar/python-selenium-bdd
https://api.github.com/repos/Fidano/ColdCukes
https://api.github.com/repos/stanfy/behave-rest
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java-maven-example
https://api.github.com/repos/mvallebr/BDD-java-cucumber-selenium
https://api.github.com/repos/gabrielfalcao/tornado-bdd-sample
https://api.github.com/repos/DavidSouther/tdd-angular
https://api.github.com/repos/rpfujiw/gui-auto-uat
https://api.github.com/repos/RobGibbens/BddWithXamarinUITest
https://api.github.com/repos/serhatbolsu/appium-python-bdd
https://api.github.com/repos/martinschneider/justtestlah
https://api.github.com/repos/speciphy/speciphy
https://api.github.com/repos/ashenwgt/awesome-bdd-with-cucumber

https://api.github.com/repos/rahulrathore44/SeleniumCucumber
https://api.github.com/repos/havvg/BDD-Experiment
https://api.github.com/repos/balamaci/blog-ui-bdd-testing
https://api.github.com/repos/giggio-samples/Exemplo-BDD-com-CSharp-Specflow-e-Headless
    -Browser
https://api.github.com/repos/serenity-bdd/bdd-trader
https://api.github.com/repos/coshx/docker-bdd
https://api.github.com/repos/oleg-toporkov/python-bdd-selenium
https://api.github.com/repos/Remchi/libapp
https://api.github.com/repos/sizuhiko/cakephp3-bdd-example
https://api.github.com/repos/20tab/django-bdd-toolkit
https://api.github.com/repos/colinbut/sales-order-system
https://api.github.com/repos/phodal/bdd-frameworks-compare
https://api.github.com/repos/behave-restful/behave-restful
https://api.github.com/repos/mikehale/chef-bdd
https://api.github.com/repos/jharmn/TwitterRestTests
https://api.github.com/repos/pradeepta02/python-behave-automation-framework
https://api.github.com/repos/blu-corner/etiqet
https://api.github.com/repos/sbtqa/api-factory
https://api.github.com/repos/johnnonolan/ChessBoard-Kata
https://api.github.com/repos/harlanji/gogherkit
https://api.github.com/repos/tinyerp/openobject-mirliton
https://api.github.com/repos/marcusoftnet/BDDWithSpecFlow
https://api.github.com/repos/shankybnl/selenium_BDD_framework
https://api.github.com/repos/testdrivenio/django-aloe-bdd
https://api.github.com/repos/dajudge/testee.fi
https://api.github.com/repos/katalon-studio-samples/katalon-bdd-cucumber-tests
https://api.github.com/repos/jagilpe/bdd-pact-microservices

100 repositórios com mais contribuidores:

https://api.github.com/repos/karma-runner/karma
https://api.github.com/repos/pester/Pester
https://api.github.com/repos/cucumber/aruba
https://api.github.com/repos/continuumsecurity/bdd-security
https://api.github.com/repos/machzqcq/CucumberJVMExamples
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java
https://api.github.com/repos/gauravkarvir/cucumber_testng_java
https://api.github.com/repos/acuminous/yadda
https://api.github.com/repos/jan-molak/serenity-js

https://api.github.com/repos/qmetry/qaf
https://api.github.com/repos/serenity-bdd/serenity-cucumber
https://api.github.com/repos/codegram/spinach
https://api.github.com/repos/DATA-DOG/godog
https://api.github.com/repos/calabash/x-platform-example
https://api.github.com/repos/BitBagCommerce/SyliusCmsPlugin
https://api.github.com/repos/rlisagor/freshen
https://api.github.com/repos/serenity-bdd/serenity-documentation
https://api.github.com/repos/hidroh/cucumber-api
https://api.github.com/repos/angiejones/restassured-with-cucumber-demo
https://api.github.com/repos/tychobrailleur/cucumber-jvm-examples
https://api.github.com/repos/serenity-bdd/serenity-cucumber-starter
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java-maven-example
https://api.github.com/repos/priyankshah217/AppiumCucumberTest
https://api.github.com/repos/pjlsergeant/test-bdd-cucumber-perl
https://api.github.com/repos/meadsteve/white-bread
https://api.github.com/repos/salad/salad
https://api.github.com/repos/behave/behave-django
https://api.github.com/repos/rahulrathore44/SeleniumCucumber
https://api.github.com/repos/mvallebr/BDD-java-cucumber-selenium
https://api.github.com/repos/ashenwgt/awesome-bdd-with-cucumber
https://api.github.com/repos/angiejones/automation-framework
https://api.github.com/repos/cpp-testing/GUnit
https://api.github.com/repos/paulgibbs/behat-wordpress-extension
https://api.github.com/repos/sorin-costea/bdd
https://api.github.com/repos/Halleck45/BDDWizard
https://api.github.com/repos/colinbut/sales-order-system
https://api.github.com/repos/serhatbolsu/appium-python-bdd
https://api.github.com/repos/larryg01/klassi-js
https://api.github.com/repos/Shashikant86/cucumber-appium
https://api.github.com/repos/serenity-bdd/bdd-trader
https://api.github.com/repos/ekino/veggies
https://api.github.com/repos/dcypherthis/wdio-boilerplate-cucumber
https://api.github.com/repos/Rhoynar/python-selenium-bdd
https://api.github.com/repos/serenity-js/tutorial-from-scripts-to-serenity
https://api.github.com/repos/cabbage-ex/cabbage
https://api.github.com/repos/stanfy/behave-rest
https://api.github.com/repos/katalon-studio-samples/katalon-bdd-cucumber-tests
https://api.github.com/repos/ukrvassabi/serenity-bdd-test
https://api.github.com/repos/everzet/fullstack-bdd-sflive2012

https://api.github.com/repos/graze/cucumber-rest-bdd
https://api.github.com/repos/CodemateLtd/Android-Cucumber-BDD-Sample
https://api.github.com/repos/pradeepta02/python-behave-automation-framework
https://api.github.com/repos/Chorus-bdd/Chorus
https://api.github.com/repos/mikek/behave-http
https://api.github.com/repos/Galad/tranquire
https://api.github.com/repos/AndyLPK247/behavior-driven-python
https://api.github.com/repos/Frameworkium/frameworkium-bdd
https://api.github.com/repos/blu-corner/etiqet
https://api.github.com/repos/PredixDev/predix-bdd-cucumber
https://api.github.com/repos/ttutisani/Xunit.Gherkin.Quick
https://api.github.com/repos/Fidano/ColdCukes
https://api.github.com/repos/RobGibbens/BddWithXamarinUITest
https://api.github.com/repos/hdorgeval/cucumber-ts-starter
https://api.github.com/repos/shankybnl/selenium_BDD_framework
https://api.github.com/repos/ing-bank/bdd-mobile-security-automation-framework
https://api.github.com/repos/serenity-dojo/caffeinate-me
https://api.github.com/repos/sitture/cucumber-jvm-extentreport
https://api.github.com/repos/noblesamurai/cucumis
https://api.github.com/repos/bmsantos/cola-tests
https://api.github.com/repos/balamaci/blog-ui-bdd-testing
https://api.github.com/repos/oleg-toporkov/python-bdd-selenium
https://api.github.com/repos/jharmn/TwitterRestTests
https://api.github.com/repos/johnnonolan/ChessBoard-Kata
https://api.github.com/repos/readyapi/testserver-cucumber
https://api.github.com/repos/navalkgupta/protractor-cucumber-tests
https://api.github.com/repos/martinschneider/justtestlah
https://api.github.com/repos/serenity-bdd/serenity-exercises
https://api.github.com/repos/DanteLore/bdd-spark
https://api.github.com/repos/serenity-js/seed-cucumber
https://api.github.com/repos/yoshuawuyts/cuke-tap
https://api.github.com/repos/coshx/docker-bdd
https://api.github.com/repos/20tab/django-bdd-toolkit
https://api.github.com/repos/olegpidsadnyi/pytest-bdd-example
https://api.github.com/repos/testdrivenio/django-aloe-bdd
https://api.github.com/repos/sabre1041/ose-bdd-demo
https://api.github.com/repos/mica16/BDD-TDD-Demo
https://api.github.com/repos/raghuk9/PhotonLabs_AutomationFramework_BDD_TDMS_MDD
https://api.github.com/repos/the-creative-tester/python-behave-web-browser-automation
    -example

https://api.github.com/repos/behave-restful/behave-restful
https://api.github.com/repos/julialiuliu/Appium-BDD-python-mobile-app-testing
https://api.github.com/repos/serenity-bdd/serenity-cli
https://api.github.com/repos/haroon-sheikh/cucumber-appium-ruby-example
https://api.github.com/repos/tobiasheine/Movies
https://api.github.com/repos/bbqsrc/cucumber-rust
https://api.github.com/repos/Remchi/libapp
https://api.github.com/repos/harlanji/gogherkit
https://api.github.com/repos/sauliuz/nightmare-cucumberjs
https://api.github.com/repos/Qualitestuk/SerenityBDD-Appium-Cross-Platform-Testing
    -Solution
https://api.github.com/repos/spyoungtech/behave-webdriver
https://api.github.com/repos/Xceptance/neodymium-library

100 repositórios mais atuais:

https://api.github.com/repos/spyoungtech/behave-webdriver
https://api.github.com/repos/Rhoynar/python-selenium-bdd
https://api.github.com/repos/haroon-sheikh/cucumber-appium-ruby-example
https://api.github.com/repos/colinbut/sales-order-system
https://api.github.com/repos/graze/cucumber-rest-bdd
https://api.github.com/repos/bbqsrc/cucumber-rust
https://api.github.com/repos/AndyLPK247/behavior-driven-python
https://api.github.com/repos/karma-runner/karma
https://api.github.com/repos/DATA-DOG/godog
https://api.github.com/repos/BitBagCommerce/SyliusCmsPlugin
https://api.github.com/repos/serenity-bdd/serenity-cucumber-starter
https://api.github.com/repos/cucumber/aruba
https://api.github.com/repos/gauravkarvir/cucumber_testng_java
https://api.github.com/repos/pester/Pester
https://api.github.com/repos/continuumsecurity/bdd-security
https://api.github.com/repos/jan-molak/serenity-js
https://api.github.com/repos/salad/salad
https://api.github.com/repos/dariusz-wozniak/TddBook-Code
https://api.github.com/repos/paulgibbs/behat-wordpress-extension
https://api.github.com/repos/angiejones/restassured-with-cucumber-demo
https://api.github.com/repos/machzqcq/CucumberJVMExamples
https://api.github.com/repos/cpp-testing/GUnit
https://api.github.com/repos/martinschneider/justtestlah
https://api.github.com/repos/thiagodp/concordialang

https://api.github.com/repos/blu-corner/etiqet
https://api.github.com/repos/behave/behave-django
https://api.github.com/repos/Chorus-bdd/Chorus
https://api.github.com/repos/cedaniel200/base_project_for_web_automation_projects
https://api.github.com/repos/qmetry/qaf
https://api.github.com/repos/ing-bank/bdd-mobile-security-automation-framework
https://api.github.com/repos/mica16/BDD-TDD-Demo
https://api.github.com/repos/ttutisani/Xunit.Gherkin.Quick
https://api.github.com/repos/serenity-bdd/bdd-trader
https://api.github.com/repos/cabbage-ex/cabbage
https://api.github.com/repos/hidroh/cucumber-api
https://api.github.com/repos/julialiuliu/Appium-BDD-python-mobile-app-testing
https://api.github.com/repos/ekino/veggies
https://api.github.com/repos/Galad/tranquire
https://api.github.com/repos/ashenwgt/awesome-bdd-with-cucumber
https://api.github.com/repos/hdorgeval/cucumber-ts-starter
https://api.github.com/repos/phodal/bdd-frameworks-compare
https://api.github.com/repos/daniel-hall/TestKit
https://api.github.com/repos/acuminous/yadda
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java
https://api.github.com/repos/tobiasheine/Movies
https://api.github.com/repos/sbtqa/api-factory
https://api.github.com/repos/anair-it/drools-cucumber
https://api.github.com/repos/noblesamurai/cucumis
https://api.github.com/repos/shaunc/cucumber_nim
https://api.github.com/repos/Xceptance/neodymium-library
https://api.github.com/repos/serenity-bdd/serenity-cucumber
https://api.github.com/repos/serenity-bdd/serenity-documentation
https://api.github.com/repos/lucascourot/PHPKataTrainReservation
https://api.github.com/repos/calabash/x-platform-example
https://api.github.com/repos/angiejones/automation-framework
https://api.github.com/repos/giggio-samples/Exemplo-BDD-com-CSharp-Specflow-e-Headless
    -Browser
https://api.github.com/repos/meadsteve/white-bread
https://api.github.com/repos/DanteLore/bdd-spark
https://api.github.com/repos/giggio-samples/outro-exemplo-de-bdd-com-csharp
https://api.github.com/repos/giggio-samples/Exemplo-Blog-BDD-CSharp
https://api.github.com/repos/codegram/spinach
https://api.github.com/repos/pjlsergeant/test-bdd-cucumber-perl
https://api.github.com/repos/selenium-cucumber/selenium-cucumber-java-maven-example

https://api.github.com/repos/larryg01/klassi-js
https://api.github.com/repos/serenity-bdd/serenity-cli
https://api.github.com/repos/serhatbolsu/appium-python-bdd
https://api.github.com/repos/katalon-studio-samples/katalon-bdd-cucumber-tests
https://api.github.com/repos/behave-restful/behave-restful
https://api.github.com/repos/Qualitestuk/SerenityBDD-Appium-Cross-Platform-Testing-Solution
https://api.github.com/repos/Remchi/libapp
https://api.github.com/repos/ustwo/bdd-crossplatform-apps
https://api.github.com/repos/sauliuz/nightmare-cucumberjs
https://api.github.com/repos/dajudge/testee.fi
https://api.github.com/repos/testdrivenio/django-aloe-bdd
https://api.github.com/repos/havvg/BDD-Experiment
https://api.github.com/repos/shankybnl/selenium_BDD_framework
https://api.github.com/repos/ukrvassabi/serenity-bdd-test
https://api.github.com/repos/jagilpe/bdd-pact-microservices
https://api.github.com/repos/RobGibbens/BddWithXamarinUITest
https://api.github.com/repos/erikedin/ExecutableSpecifications.jl
https://api.github.com/repos/rlisagor/freshen
https://api.github.com/repos/davidemoro/cookiecutter-qa
https://api.github.com/repos/stanfy/behave-rest
https://api.github.com/repos/llonchj/browsersteps
https://api.github.com/repos/Chorus-bdd/chorus-js
https://api.github.com/repos/spilio/apispots-bdd
https://api.github.com/repos/balamaci/blog-ui-bdd-testing
https://api.github.com/repos/sitture/cucumber-jvm-extentreport
https://api.github.com/repos/20tab/django-bdd-toolkit
https://api.github.com/repos/sorin-costea/bdd
https://api.github.com/repos/serenity-bdd/serenity-exercises
https://api.github.com/repos/js-republic/cucumber-slice-by-slice-workshop
https://api.github.com/repos/the-creative-tester/python-behave-web-browser-automation-example
https://api.github.com/repos/dcypherthis/wdio-boilerplate-cucumber
https://api.github.com/repos/CodemateLtd/Android-Cucumber-BDD-Sample
https://api.github.com/repos/jholsgrove/BDDRestSharp
https://api.github.com/repos/rahulrathore44/SeleniumCucumber
https://api.github.com/repos/sizuhiko/cakephp3-bdd-example
https://api.github.com/repos/everzet/fullstack-bdd-sflive2012
https://api.github.com/repos/skalnik/TDD-BDD-Example